Matheus C. Fernandes
fernandes@g.harvard.edu
CS181-S17

Assignment #2
Due: 5:00pm Feb 24th, 2017

Collaborators:
Bolei Deng

SOLUTION - Do Not Distribute
Homework 2: Bayesian Methods and Multiclass Classification

Introduction

This homework is about Bayesian methods and multiclass classification. In lecture we have primarily focused on binary classifiers trained to discriminate between two classes. In multiclass classification, we discriminate between three or more classes. We encourage you to first read the Bishop textbook coverage of these topic, particularly: Section 4.2 (Probabilistic Generative Models), Section 4.3 (Probabilistic Discriminative Models).

As usual, we imagine that we have the input matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$ (or perhaps they have been mapped to some basis $\mathbf{\Phi}$, without loss of generality) but our outputs are now "one-hot coded". What that means is that, if there are c output classes, then rather than representing the output label y as an integer $1, 2, \ldots, c$, we represent $\mathbf{y}$ as a binary vector of length c. 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 $C_3 = [0, 0, 1, 0, 0, 0, 0]$. If there are c classes, the set of possible outputs is $\{C_1 \ldots C_c\} = \{C_k\}_{k=1}^{c}$. Throughout the assignment we will assume that output $\mathbf{y} \in \{C_k\}_{k=1}^{c}$.

The problem set has four problems:

- In the first problem, you will explore the properties of Bayesian estimation methods for the Bernoulli model as well as the special case of Bayesian linear regression with a simple prior.
- In the second problem, you will explore the properties of the softmax function, which is central to the method of multiclass logistic regression.
- In the third problem, you will dive into matrix algebra and the methods behind generative multiclass classifications. You will extend the discrete classifiers that we see in lecture to a Gaussian model.
- Finally, in the fourth problem, you will implement logistic regression as well as a generative classifier from close to scratch.

Problem 1 (Bayesian Methods, 10 pts)

This question helps to build your understanding of the maximum-likelihood estimation (MLE) vs. maximum a posterior estimator (MAP) and posterior predictive estimator, first in the Beta-Bernoulli model and then in the linear regression setting.

First consider the Beta-Bernoulli model (and see lecture 5.)

1. Write down the expressions for the MLE, MAP and posterior predictive distributions, and for a prior $\theta \sim Beta(4, 2)$ on the parameter of the Bernoulli, and with data $D = 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0$, plot the three different estimates after each additional sample.
2. Plot the posterior distribution (prior for 0 examples) on θ after 0, 4, 8, 12 and 16 examples. (Using whatever tools you like.)
3. Interpret the differences you see between the three different estimators.

Second, consider the Bayesian Linear Regression model, with data $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, $\mathbf{x}_i \in \mathbb{R}^m$, $y_i \in \mathbb{R}$, and generative model

$$y_i \sim \mathcal{N}(\mathbf{w}^\top \mathbf{x}_i, \beta^{-1})$$

for (known) precision β (which is just the reciprocal of the variance). Given this, the likelihood of the data is $p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{w}, \beta^{-1}\mathbf{I})$. Consider the special case of an isotropic (spherical) prior on weights, with

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$$

4. Justify when you might use this prior in practice.
5. Using the method in lecture of taking logs, expanding and pushing terms that don't depend on w into a constant, and finally collecting terms and completing the square, confirm that the posterior on weights after data D is $\mathbf{w} \sim \mathcal{N}(\mathbf{w}|\mathbf{m}_n, \mathbf{S}_n)$, where

$$\mathbf{S}_n = (\alpha\mathbf{I} + \beta\mathbf{X}^\top\mathbf{X})^{-1}$$
$$\mathbf{m}_n = \beta\mathbf{S}_n\mathbf{X}^\top\mathbf{y}$$

6. Derive the special case of the MAP estimator for this problem as the isotropic prior becomes arbitrarily weak. What does the MAP estimator reduce to?
7. What did we observe in lecture about this estimator for the case where the prior is neither weak nor strong?

Solution

Problem 1.1

The expressions for MLE, MAP, and the posterior predictive distribution are as follows:

$$p(\theta|D) = \theta_{\text{MLE}} = \frac{n_1}{n_0 + n_1}$$

$$p(\theta|D) = \text{Beta}(\theta|n_1 + \alpha, n_0 + \beta) = \theta_{\text{MAP}} = \frac{\alpha + n_1 - 1}{\alpha + \beta + n_1 + n_0 - 2}$$

$$p(x = 1|D)_{Post} = \frac{\alpha + n_1}{\alpha + \beta + n_1 + n_0}$$

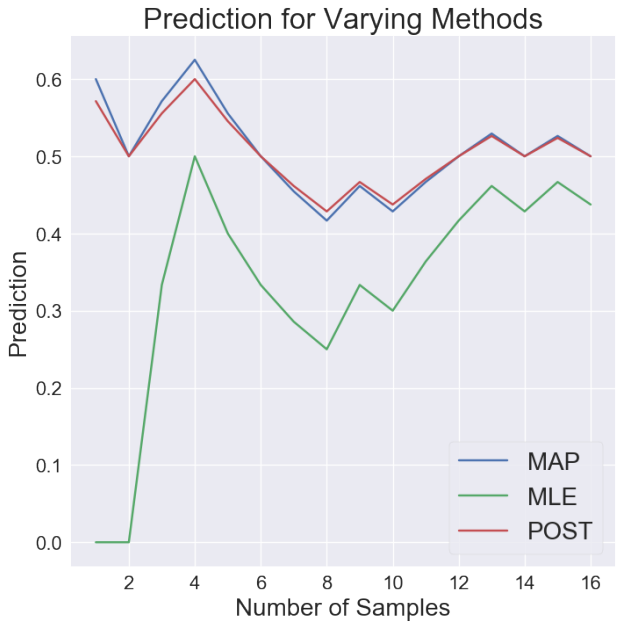

Figure 1: Plot for the three different estimates after each additional sample.

Problem 1.2

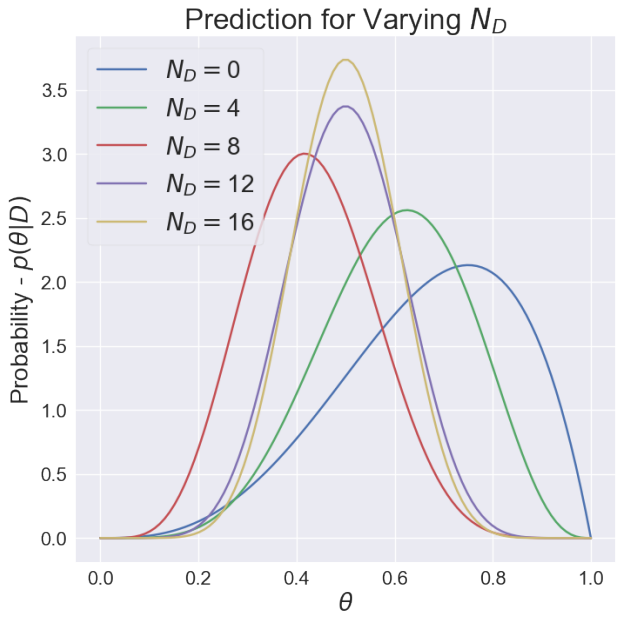

Figure 2: Posterior distribution with prirpo on θ after 0, 4, 8, 12, and 16 examples.

Problem 1.3

As we can see for the MAP and posterior estimators they follow closely to each other. On the other hand, the MLE follows a much lower path compared to MAP and the posterior estimators. This comes from the fact that we are not using a prior. Another effect of this, is that we can see that at the beginning, it is at 0, because it does not consider any distribution effects, but rather only the beginning data, in which it was all 0's. As we increase the number of samples, we also see that the MLE estimator starts converging to the other two solutions.

Problem 1.4

This prior could be useful in practice when you do not have any information on the distribution of the sample beforehand. This would be a good initial prior as it is uniform and imitates a unit, multivariate, and isotropic Gaussian distribution. Also, this does not introduce artificial bias to the model. Furthermore, it is important to note that this introduces a prior that has a mean of 0 and with independent features as the non-diagonal elements of the covariance are 0.

Problem 1.5

Given that we have the following:

$$p(\mathbf{y}|\mathbf{X},\mathbf{w}) = \mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{w},\beta^{-1}\mathbf{I})$$

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0},\alpha^{-1}\mathbf{I})$$

and Bayes rule gives us

$$p(\mathbf{w}|D) \propto p(\mathbf{w})p(\mathbf{y}|\mathbf{X},\mathbf{w})$$

such that

$$\ln(p(\mathbf{w}|D)) \propto \underbrace{\ln(p(\mathbf{w}))}_{①} + \underbrace{\ln(p(\mathbf{y}|\mathbf{X},\mathbf{w}))}_{②}.$$

$$① \Rightarrow \ln(\mathcal{N}(\mathbf{w}|\mathbf{0},\alpha^{-1}\mathbf{I}) = c - \frac{1}{2}(\mathbf{w}^\top \alpha\mathbf{I}\mathbf{w})$$

$$② \Rightarrow \ln(\mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{w},\beta^{-1}\mathbf{I})) = c - \frac{1}{2}(\mathbf{y}-\mathbf{X}\mathbf{w})^\top\beta\mathbf{I}(\mathbf{y}-\mathbf{X}\mathbf{w}) = c - \frac{\beta}{2}(\cancel{\mathbf{y}^\top\mathbf{y}}^{\text{constant}} - 2\mathbf{y}^\top\mathbf{X}\mathbf{w} + \mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w})$$

Putting ① and ② together we obtain

$$\ln(p(\mathbf{w}|D)) \propto c - \frac{\alpha\mathbf{I}}{2}(\mathbf{w}^\top\mathbf{w}) - \frac{\beta}{2}(-2\mathbf{y}^\top\mathbf{X}\mathbf{w} + \mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w})$$

$$\ln(p(\mathbf{w}|D)) \propto c - \frac{1}{2}(\alpha\mathbf{I}\mathbf{w}^\top\mathbf{w} - 2\beta\mathbf{y}^\top\mathbf{X}\mathbf{w} + \beta\mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w})$$

$$\ln(p(\mathbf{w}|D)) \propto c - \frac{1}{2}(\mathbf{w}^\top(\alpha\mathbf{I} + \beta\mathbf{X}^\top\mathbf{X})\mathbf{w} - 2\beta\mathbf{y}^\top\mathbf{X}\mathbf{w})$$

which using completing the square yields the form

$$\ln(\mathcal{N}(\mathbf{w}|\mathbf{m}_n, \mathbf{S}_n)) = c - \frac{1}{2}\left(\mathbf{w}^\top \mathbf{S}_n^{-1}\mathbf{w} - 2\mathbf{m}_n^\top \mathbf{S}_n^{-1}\mathbf{w} + \mathbf{m}_n^\top \mathbf{S}_n^{-1}\mathbf{m}_n\right)$$

where

$$\begin{aligned}\mathbf{S}_n &= (\alpha\mathbf{I} + \beta\mathbf{X}^\top\mathbf{X})^{-1} \\ \mathbf{m}_n &= \beta\mathbf{S}_n\mathbf{X}^\top\mathbf{y}.\end{aligned}$$

and ultimately simplified to

$$\ln(\mathcal{N}(\mathbf{w}|\mathbf{m}_n, \mathbf{S}_n)) = c - \frac{1}{2}(\mathbf{w} - \mathbf{m}_n)^\top \mathbf{S}_n^{-1}(\mathbf{w} - \mathbf{m}_n).$$

■

Problem 1.6

If the isotropic prior becomes arbitrarily weak, then $\alpha \to 0$, meaning that the MAP estimator becomes

$$\ln(\mathcal{N}(\mathbf{w}|\mathbf{m}_n, \mathbf{S}_n)) = c - \frac{1}{2}(\mathbf{w} - \mathbf{m}_n)^\top \mathbf{S}_n^{-1}(\mathbf{w} - \mathbf{m}_n)$$

where

$$\begin{aligned}\mathbf{S}_n &= (\beta\mathbf{X}^\top\mathbf{X})^{-1} \\ \mathbf{m}_n &= \beta\mathbf{S}_n\mathbf{X}^\top\mathbf{y}.\end{aligned}$$

which is simply the MLE distribution.

Problem 1.7

If the prior is neither weak nor strong, then we see that it behaves moderately as a regularization of the model and we recover essentially the ridge regression once again. This can be seen by bringing together the α and the β into one fractional parameter. This is really useful, as we can see the similarities between the functions of the different methods.

End Solution

Problem 2 (Properties of Softmax, 8pts)

We have explored logistic regression, which is a discriminative probabilistic model over two classes. For each input $\mathbf{x}$, logistic regression outputs a probability of the class output y using the logistic sigmoid function.

The softmax transformation is an important generalization of the logistic sigmoid to the case of c classes. It takes as input a vector, and outputs a transformed vector of the same size,

$$\text{softmax}(\mathbf{z})_k = \frac{\exp(z_k)}{\sum_{\ell=1}^{c} \exp(z_\ell)}, \quad \text{for all } k$$

Multiclass logistic regression uses the softmax transformation over vectors of size c. Let $\{\mathbf{w}_\ell\} = \{\mathbf{w}_1 \dots \mathbf{w}_c\}$ denote the parameter vectors for each class. In particular, multiclass logistic regression defines the probability of class k as,

$$p(\mathbf{y} = C_k \mid \mathbf{x}; \{\mathbf{w}_\ell\}) = \text{softmax}([\mathbf{w}_1^\top \mathbf{x} \dots \mathbf{w}_c^\top \mathbf{x}]^\top)_k = \frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_{\ell=1}^{c} \exp(\mathbf{w}_\ell^\top \mathbf{x})} .$$

As above, we are using $\mathbf{y} = C_k$ to indicate the output vector that represents class k.

Assuming data $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, the negated log-likelihood can be written in the standard form, as

$$\mathcal{L}(\{\mathbf{w}_\ell\}) = -\sum_{i=1}^{n} \ln p(\mathbf{y}_i \mid \mathbf{x}_i; \{\mathbf{w}_\ell\})$$

Softmax is an important function in the context of machine learning, and you will see it again in other models, such as neural networks. In this problem, we aim to gain intuitions into the properties of softmax and multiclass logistic regression.

Show that:

1. The output of the softmax function is a vector with non-negative components that are at most 1.
2. The output of the softmax function defines a distribution, so that in addition, the components sum to 1.
3. Softmax preserves order. This means that if elements $z_k < z_\ell$, in $\mathbf{z}$, then $\text{softmax}(\mathbf{z})_k < \text{softmax}(\mathbf{z})_\ell$ for any k, ℓ.
4. Show that
$$\frac{\partial \text{softmax}(\mathbf{z})_k}{\partial z_j} = \text{softmax}(\mathbf{z})_k (I_{kj} - \text{softmax}(\mathbf{z})_j) \quad \text{for any } k, j$$
, where indicator $I_{kj} = 1$ if $k = j$ and $I_{kj} = 0$ otherwise.
5. Using your answer to the previous question, show that
$$\frac{\partial}{\partial \mathbf{w}_k} \mathcal{L}(\{\mathbf{w}_\ell\}) = \sum_{i=1}^{n} (p(\mathbf{y}_i = C_k | \mathbf{x}_i; \{\mathbf{w}_\ell\}) - y_{ik}) \mathbf{x}_i$$
By the way, this may be useful for Problem 3!

Solution

Problem 2.1

We know that $e^x > 1$ for $x \in \mathbb{R}$ and that

$$\sum_{l=1}^{c} \exp z_l \geq \exp z_k \Rightarrow \frac{\exp z_k}{\sum_{l=1}^{c} \exp z_l} \leq 1.$$

This means that the softmax function is a vector with non-negative components that are at most 1.

■

Problem 2.2

$$\boxed{\sum_{k=1}^{c} \text{softmax}(\mathbf{z})_k = \frac{\sum_{k=1}^{c} \exp z_k}{\sum_{l=1}^{c} \exp z_l} = 1}$$

■

Problem 2.3

If $z_k < z_l$, then

$$\exp(z_k) < \exp(z_l).$$

Also,

$$\text{softmax}(\mathbf{z})_k = \frac{\exp z_k}{\sum_{i=1}^{c} \exp z_i} \quad \text{and} \quad \text{softmax}(\mathbf{z})_l = \frac{\exp z_l}{\sum_{i=1}^{c} \exp z_i}$$

then,

$$\frac{\exp z_k}{\sum_{i=1}^{c} \exp z_i} < \frac{\exp z_l}{\sum_{i=1}^{c} \exp z_i}.$$

Thus, this ultimately means that

$$\boxed{\text{softmax}(\mathbf{z})_k < \text{softmax}(\mathbf{z})_l}$$

■

Problem 2.4

$$\begin{aligned}\frac{\partial \text{softmax}(\mathbf{z})_k}{\partial z_j} &= \frac{\partial}{\partial z_j}\left(\frac{\exp z_k}{\sum_{l=1}^{c}\exp z_l}\right) \\ &= \frac{\partial}{\partial z_j}\left(\exp z_k \frac{1}{\sum_{l=1}^{c}\exp z_l}\right) \\ &= \underbrace{\frac{\partial \exp z_k}{\partial z_j}\cdot\frac{1}{\sum_{l=1}^{c}\exp z_l}}_{①} + \underbrace{\exp z_k \cdot \frac{\partial}{\partial z_j}\left(\frac{1}{\sum_{l=1}^{c}\exp z_l}\right)}_{②}\end{aligned}$$

If we now take the individual derivatives, in which ② we use the chain rule, we get

$$\begin{aligned}① &= \mathbf{I}_{kj}\frac{\exp z_k}{\sum_{l=1}^{c}\exp z_l} \\ ② &= \exp z_k \frac{\partial}{\partial u}\frac{1}{u} \quad \text{where} \quad u = \left(\sum_{l=1}^{c}\exp z_l\right) \quad \text{and} \quad \frac{\partial}{\partial u_j} = \exp z_j \\ &= -\exp z_k \frac{1}{(\sum_{l=1}^{c}\exp z_k)^2}\exp z_j\end{aligned}$$

Putting ① and ② together, this simplifies to

$$\boxed{\text{softmax}(\mathbf{z})_k(\mathbf{I}_{kj} - \text{softmax}(\mathbf{z})_j)}$$

■

Problem 2.5

$$p(\mathbf{y}_i|\mathbf{x}_i, \{\mathbf{w}_{\boldsymbol{\ell}}\}) = \prod_{k=1}^{c} p(\mathbf{y}_i = C_k|\mathbf{x}_i\{\mathbf{w}_{\boldsymbol{\ell}}\})$$

$$\begin{aligned}\mathcal{L}(\{\mathbf{w}_{\boldsymbol{\ell}}\}) &= -\ln\sum_{i=1}^{N} p(\mathbf{y}_i|\mathbf{x}_i, \{\mathbf{w}_{\boldsymbol{\ell}}\}) \\ &= -\sum_{i=1}^{N}\sum_{j=1}^{c}\ln\left(\prod_{k=1}^{c} p(\mathbf{y}_i = C_k|\mathbf{x}_i\{\mathbf{w}_{\boldsymbol{\ell}}\})\right)^{y_{ij}} \\ &= -\sum_{i=1}^{N}\sum_{j=1}^{c} y_{ij}\ln(\text{softmax}([\mathbf{w}_1^\top\mathbf{x}...\mathbf{w}_c^\top\mathbf{x}]^\top)_j)\end{aligned}$$

Taking the derivative w.r.t $\mathbf{w}_k$ we see that

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{w}_k}\left(\mathcal{L}(\{\mathbf{w}_{\ell}\})\right) &= -\sum_{i=1}^{N}\sum_{j=1}^{c}\frac{\partial}{\partial \mathbf{w}_k}\left(y_{ij}\ln(\text{softmax}([\mathbf{w}_1^\top \mathbf{x}...\mathbf{w}_c^\top \mathbf{x}]^\top)_j)\right) \\
&= -\sum_{i=1}^{N}\sum_{j=1}^{c} y_{ij}\frac{\partial[\mathbf{w}_1^\top \mathbf{x}...\mathbf{w}_c^\top \mathbf{x}]^\top}{\partial \mathbf{w}_k}\frac{\text{softmax}([\mathbf{w}_1^\top \mathbf{x}...\mathbf{w}_c^\top \mathbf{x}]^\top)_j(\mathbf{I}_{jk}-\text{softmax}([\mathbf{w}_1^\top \mathbf{x}...\mathbf{w}_c^\top \mathbf{x}]^\top)_k)}{\text{softmax}([\mathbf{w}_1^\top \mathbf{x}...\mathbf{w}_c^\top \mathbf{x}]^\top)_j} \\
&= -\sum_{i=1}^{N}\sum_{j=1}^{c}\mathbf{x}_i y_{ij}(\mathbf{I}_{jk}-\text{softmax}([\mathbf{w}_1^\top \mathbf{x}...\mathbf{w}_c^\top \mathbf{x}]^\top)_k) \\
&= -\sum_{i=1}^{N}\sum_{j=1}^{c}\mathbf{x}_i y_{ij}\mathbf{I}_{jk} - \mathbf{x}_i y_{ij}\text{softmax}([\mathbf{w}_1^\top \mathbf{x}...\mathbf{w}_c^\top \mathbf{x}]^\top)_k \\
&= -\sum_{i=1}^{N}\sum_{j=1}^{c}\mathbf{x}_i y_{ij}\mathbf{I}_{jk} + \sum_{i=1}^{N}\sum_{j=1}^{c}\mathbf{x}_i y_{ij}\text{softmax}([\mathbf{w}_1^\top \mathbf{x}...\mathbf{w}_c^\top \mathbf{x}]^\top)_k \\
&= -\sum_{i=1}^{N}\mathbf{x}_i y_{ik} + \sum_{i=1}^{N}\mathbf{x}_i\text{softmax}([\mathbf{w}_1^\top \mathbf{x}...\mathbf{w}_c^\top \mathbf{x}]^\top)_k \quad \text{where all i}^{th}\text{ row of } y_{ij} \text{ will sum to 1} \\
&= \sum_{i=1}^{N} x_i(\text{softmax}([\mathbf{w}_1^\top \mathbf{x}...\mathbf{w}_c^\top \mathbf{x}]^\top)_k - y_{ik}) \\
&= \boxed{\sum_{i=1}^{N}(p(\mathbf{y}_i|\mathbf{x}_i,\{\mathbf{w}_{\ell}\}) - y_{ik})x_i}
\end{aligned}$$

■

End Solution

Problem 3 (Return of matrix calculus, 10pts)

Consider now a generative c-class model. We adopt class prior $p(\mathbf{y} = C_k; \boldsymbol{\pi}) = \pi_k$ for all $k \in \{1, \ldots, c\}$ (where π_k is a parameter of the prior). Let $p(\mathbf{x}|\mathbf{y} = C_k)$ denote the class-conditional density of features $\mathbf{x}$ (in this case for class C_k). Consider the data set $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ where as above $\mathbf{y}_i \in \{C_k\}_{k=1}^c$ is encoded as a one-hot target vector.

1. Write out the negated log-likelihood of the data set, $-\ln p(D; \boldsymbol{\pi})$.
2. Since the prior forms a distribution, it has the constraint that $\sum_k \pi_k - 1 = 0$. Using the hint on Lagrange multipliers below, give the expression for the maximum-likelihood estimator for the prior class-membership probabilities, i.e. $\hat{\pi}_k$. 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!

For the remaining questions, let the class-conditional probabilities be Gaussian distributions with the same covariance matrix

$$p(\mathbf{x}|\mathbf{y} = C_k) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}), \text{ for } k \in \{1, \ldots, c\}$$

and different means $\boldsymbol{\mu}_k$ for each class.

3. Derive the gradient of the negative log-likelihood with respect to vector $\boldsymbol{\mu}_k$. 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. Derive the maximum-likelihood estimator for vector $\boldsymbol{\mu}_k$. Once again, your final answer should seem intuitive.
5. Derive the gradient for the negative log-likelihood with respect to the covariance matrix $\boldsymbol{\Sigma}$ (i.e., looking to find an MLE for the covariance). Since you are differentiating with respect to a *matrix*, the resulting expression should be a matrix!
6. Derive the maximum likelihood estimator of the covariance matrix.

[Hint: Lagrange Multipliers. Lagrange Multipliers are a method for optimizing a function f with respect to an equality constraint, i.e.

$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ s.t. } g(\mathbf{x}) = 0.$$

This can be turned into an unconstrained problem by introducing a Lagrange multiplier λ and constructing the Lagrangian function,

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x}).$$

It can be shown that it is a necessary condition that the optimum is a critical point of this new function. We can find this point by solving two equations:

$$\frac{\partial L(\mathbf{x}, \lambda)}{\partial \mathbf{x}} = 0 \text{ and } \frac{\partial L(\mathbf{x}, \lambda)}{\partial \lambda} = 0$$

Cookbook formulas. Here are some formulas you might want to consider using to compute difficult gradients. You can use them 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: $\mathbf{X}^{-\top} := (\mathbf{X}^\top)^{-1}$

$$\frac{\partial \mathbf{a}^\top \mathbf{X}^{-1} \mathbf{b}}{\partial \mathbf{X}} = -\mathbf{X}^{-\top} \mathbf{a} \mathbf{b}^\top \mathbf{X}^{-\top}$$

$$\frac{\partial \ln |\det(\mathbf{X})|}{\partial \mathbf{X}} = \mathbf{X}^{-\top}$$

Solution

Problem 3.1

If we want to find the negative log likelihood of the provided data set ($-\ln p(D;\boldsymbol{\pi}) = -\ln p(\{\mathbf{x}_i, \mathbf{y}_i\};\boldsymbol{\pi})$, which is a joint distribution), then we must recall

$$\prod_{i=1}^{n} p(\mathbf{x}_i, \mathbf{y}_i;\boldsymbol{\pi}) = \prod_{i=1}^{n} p(\mathbf{y}_i = C_k;\boldsymbol{\pi}) p(\mathbf{x}_i|\mathbf{y}_i = C_k)$$

Thus, to obtain the likelihood we multiply it out for each term

$$p(\mathbf{y}_i = C_k;\boldsymbol{\pi}) = \pi_k$$

$$\begin{aligned} -\ln p(\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{n}, \boldsymbol{\pi}) &= -\ln \prod_{i=1}^{n}\prod_{k=1}^{c} \pi_k^{y_{ik}} \\ &= -\sum_{i=1}^{n}\sum_{k=1}^{c} y_{ik} \ln \pi_k \end{aligned}$$

Such that at the end, because of the ln's we obtain the sum of the prior and the posterior

$$\boxed{-\sum_{i=1}^{n}\sum_{k=1}^{c} y_{ik}(\ln(\pi_k) + \ln(p(\mathbf{x}_i|\mathbf{y}_i = C_k)))}$$

Problem 3.2

To obtain the expression for the maximum-likelihood estimation for the prior class membership we must minimize the negative log likelihood expressed above as

$$\min_{\boldsymbol{\pi}} -\sum_{i=1}^{n}\sum_{k=1}^{c} y_{ik} \ln \pi_k$$

with the constraint

$$\sum_{k=1}^{c} \pi_c - 1 = 0.$$

We must solve the coupled equations using the Lagrange multiplier approach. To do so we end up with the equation

$$L(\boldsymbol{\pi}, \lambda) = -\sum_{i=1}^{n}\sum_{k=1}^{c} y_{ik} \ln \pi_k + \lambda \left(\sum_{k=1}^{c} \pi_k - 1 \right).$$

Now, as according to the Lagrange Multipliers, we need to optimize w.r.t. $\boldsymbol{\pi}$ and also w.r.t. λ.

$$\frac{\partial L(\boldsymbol{\pi}, \lambda)}{\partial \boldsymbol{\pi}_i} = -\sum_{i=1}^{n} \frac{y_{ij}}{\pi_k} - \lambda = 0 \quad \Rightarrow \quad \pi_k = \sum_{i=1}^{n} \frac{y_{ik}}{\lambda}$$

$$\frac{\partial L(\boldsymbol{\pi}, \lambda)}{\partial \boldsymbol{\pi}_i} = \sum_{k=1}^{c} \pi_k - 1 = 0 \quad \Rightarrow \quad \sum_{k=1}^{c} \pi_k = 1$$

Putting the above 2 together we obtain

$$\boxed{\hat{\pi}_k = \frac{\sum_{i=1}^{n} y_{ik}}{\sum_{k=1}^{c}\sum_{i=1}^{n} y_{ik}} = \frac{\mathbf{Y}^\top \mathbf{1}}{\mathbf{1Y1}}}$$

Problem 3.3

So the negative likelihood is given as

$$\begin{aligned}
-\ln \prod_{i=1}^{n}\prod_{k=1}^{c} p(\mathbf{x}_i|y_i = C_k)^{y_{ik}} &= \sum_{i=1}^{n}\sum_{k=1}^{c} -\ln(\mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}))^{y_{ik}} \\
&= \sum_{i=1}^{n}\sum_{k=1}^{c} -y_{ik} \ln\left((2\pi)^{-1/2}(|\boldsymbol{\Sigma}|)^{-1/2}\exp\left(-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_i)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_i)\right)\right) \\
&= \sum_{i=1}^{n}\sum_{k=1}^{c} \left(\frac{1}{2}\ln(2\pi) + \frac{1}{2}\ln(|\boldsymbol{\Sigma}|) + \frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_i)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_i)\right) y_{ik} \\
&= \sum_{i=1}^{n}\sum_{k=1}^{c} \left(c + \frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_i)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_i)\right) y_{ik}
\end{aligned}$$

Taking the gradient w.r.t. $\boldsymbol{\mu}_k$

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\mu}_k}\left(-\ln \prod_{i=1}^{n}\prod_{k=1}^{c} p(\mathbf{x}_i|y_i = C_k)^{y_{ik}}\right) &= \sum_{i=1}^{n}\sum_{k=1}^{c} \frac{\partial}{\partial \boldsymbol{\mu}_i}\left(c + \frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_i)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_i)\right) y_{ik} \\
&= \frac{1}{2}\sum_{i=1}^{n} -(\boldsymbol{\Sigma}^{-1} + \boldsymbol{\Sigma}^{-\top})(\mathbf{x}_i - \boldsymbol{\mu}_i))y_{ik} \quad \text{using chaing rule} \\
&= \frac{1}{2}\sum_{i=1}^{n} -(2\boldsymbol{\Sigma}^{-1})(\mathbf{x}_i - \boldsymbol{\mu}_i))y_{ik} \\
&= \boldsymbol{\Sigma}^{-1}\sum_{i=1}^{n} -(\mathbf{x}_i - \boldsymbol{\mu}_i))y_{ik} \\
&= \boldsymbol{\Sigma}^{-1}\left(\sum_{i=1}^{n} \mathbf{x}_i y_{ik} - \boldsymbol{\mu}_i \sum_{i=1}^{n} y_{ik}\right)
\end{aligned}$$

Problem 3.4

Here we must solve for the gradient above and set it to 0 to obtain the estimator

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{\mu}_k}\left(-\ln\prod_{i=1}^{n}\prod_{k=1}^{c} p(\mathbf{x}_i|y_i=C_k)^{y_{ik}}\right) =& 0\\ =& \boldsymbol{\Sigma}^{-1}\left(\sum_{i=1}^{n}\mathbf{x}_i y_{ik} - \boldsymbol{\mu}_i\sum_{i=1}^{n} y_{ik}\right)\\ =& \sum_{i=1}^{n}\mathbf{x}_i y_{ik} - \boldsymbol{\mu}_i \cancel{\sum_{i=1}^{n} y_{ik}}^{n_k}\end{aligned}$$

$$\boxed{\hat{\boldsymbol{\mu}}_k = \frac{1}{n_k}\sum_{i=1}^{n}\mathbf{x}_i y_{ik}}$$

Problem 3.5

From the previous problem we know that

$$-\ln\prod_{i=1}^{n}\prod_{k=1}^{c} p(\mathbf{x}_i|y_i=C_k)^{y_{ik}} = \sum_{i=1}^{n}\sum_{k=1}^{c}\left(\frac{1}{2}\ln(2\pi) + \frac{1}{2}\ln(|\boldsymbol{\Sigma}|) + \frac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu}_i)^\top\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_i)\right) y_{ik}$$

Now if we take the gradient of this w.r.t. $\boldsymbol{\Sigma}$ using the formulas from the cookbook we get

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{\Sigma}}\left(-\ln\prod_{i=1}^{n}\prod_{k=1}^{c} p(\mathbf{x}_i|y_i=C_k)^{y_{ik}}\right) =& \sum_{i=1}^{n}\sum_{k=1}^{c}\frac{\partial}{\partial \boldsymbol{\Sigma}}\left(\frac{1}{2}\ln(2\pi) + \frac{1}{2}\ln(|\boldsymbol{\Sigma}|) + \frac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu}_i)^\top\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_i)\right) y_{ik}\\ =& \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{c}\left(\boldsymbol{\Sigma}^{-\top} - \boldsymbol{\Sigma}^{-\top}(\mathbf{x}_i-\boldsymbol{\mu}_i)(\mathbf{x}_i-\boldsymbol{\mu}_i)^\top\boldsymbol{\Sigma}^{-\top}\right) y_{ik}\\ =& \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{c}\boldsymbol{\Sigma}^{-1}\left(\mathbf{I} - \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_i)(\mathbf{x}_i-\boldsymbol{\mu}_i)^\top\right) y_{ik}\\ =& \frac{1}{2}\boldsymbol{\Sigma}^{-1}\sum_{i=1}^{n}\sum_{k=1}^{c}\left(\mathbf{I} - \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_i)(\mathbf{x}_i-\boldsymbol{\mu}_i)^\top\right) y_{ik}\end{aligned}$$

Problem 3.6

To derive the maximum likelihood, we take the derivative and set it to zero and solve for $\mathbf{\Sigma}$

$$\begin{aligned}\frac{\partial}{\partial \mathbf{\Sigma}}\left(-\ln \prod_{i=1}^{n}\prod_{k=1}^{c} p(\mathbf{x}_i|y_i=C_k)^{y_{ik}}\right) =& 0 \\ =& \frac{1}{2}\mathbf{\Sigma}^{-1}\sum_{i=1}^{n}\sum_{k=1}^{c}\left(\mathbf{I}-\mathbf{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_i)(\mathbf{x}_i-\boldsymbol{\mu}_i)^\top\right)y_{ik} \\ =& \sum_{i=1}^{n}\sum_{k=1}^{c}\left(\mathbf{I}-\mathbf{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_i)(\mathbf{x}_i-\boldsymbol{\mu}_i)^\top\right)y_{ik}\end{aligned}$$

Where we can now continue solving for $\mathbf{\Sigma}$

$$\sum_{i=1}^{n}\sum_{k=1}^{c}\mathbf{I}y_{ik} = \mathbf{\Sigma}^{-1}\sum_{i=1}^{n}\sum_{k=1}^{c}(\mathbf{x}_i-\boldsymbol{\mu}_i)(\mathbf{x}_i-\boldsymbol{\mu}_i)^\top y_{ik}$$

$$\mathbf{I}\cancel{\sum_{i=1}^{n}\sum_{k=1}^{c}}^{n_i} y_{ik} = \mathbf{\Sigma}^{-1}\sum_{i=1}^{n}\sum_{k=1}^{c}(\mathbf{x}_i-\boldsymbol{\mu}_i)(\mathbf{x}_i-\boldsymbol{\mu}_i)^\top y_{ik}$$

$$\boxed{\mathbf{\Sigma} = \frac{1}{n_i}\sum_{i=1}^{n}\sum_{k=1}^{c}(\mathbf{x}_i-\boldsymbol{\mu}_i)(\mathbf{x}_i-\boldsymbol{\mu}_i)^\top y_{ik}}$$

End Solution

4. Classifying Fruit [15pts]

You're tasked with classifying three different kinds of fruit, based on their heights and widths. Figure 3 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
...
```

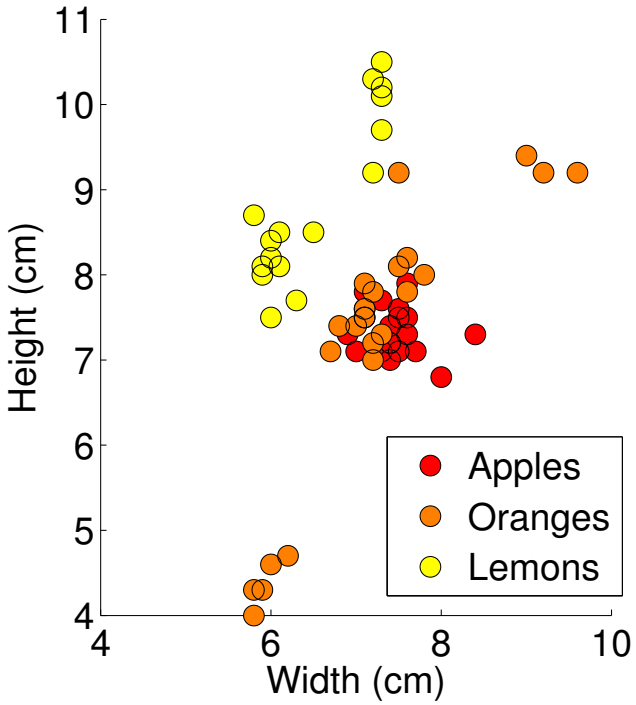

Figure 3: Heights and widths of apples, oranges, and lemons. These fruit were purchased and measured by Iain Murray: http://homepages.inf.ed.ac.uk/imurray2/teaching/oranges_and_lemons/.

Problem 4 (Classifying Fruit, 15pts)

You should implement the following:

- 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 negative log likelihood. You will need to find good values for the learning rate η and regularization strength λ.
- A generative classifier with Gaussian class-conditional densities, as in Problem 3. 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. 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 on Piazza. In general, linear algebra and random variable functions are fine. The controller file is `problem4.py`, in which you will specify hyperparameters. The actual implementations you will write will be in `LogisticRegression.py` and `GaussianGenerativeModel.py`.

You will be given 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. You should include these in this assignment.
- Which classifiers model the distributions well?
- What explains the differences?

In addition to comparing the decision boundaries of the three models visually:

- For logistic regression, report negative log-likelihood loss for several configurations of hyperparameters. Why are your final choices of learning rate (η) and regularization strength (λ) reasonable? Plot loss during training for the best of these configurations, with iterations on the x-axis and loss on the y-axis (one way to do this is to add a method to the LogisticRegression Class that displays loss).
- For both Gaussian generative models, report likelihood. In the separate covariance matrix case, be sure to use the covariance matrix that matches the true class of each data point.

Solution

The `visualize()` ***method for each classifier will save a plot that will show the decision boundaries. You should include these in this assignment.***

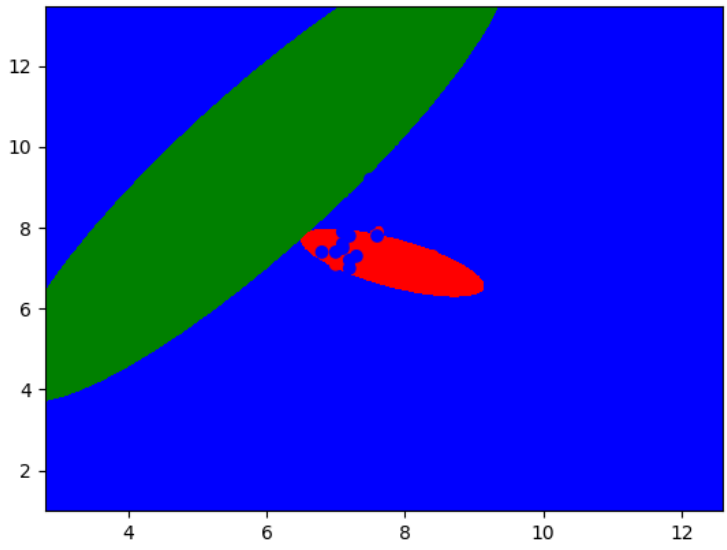

Figure 4: Separate Covariances Generative Model Plot

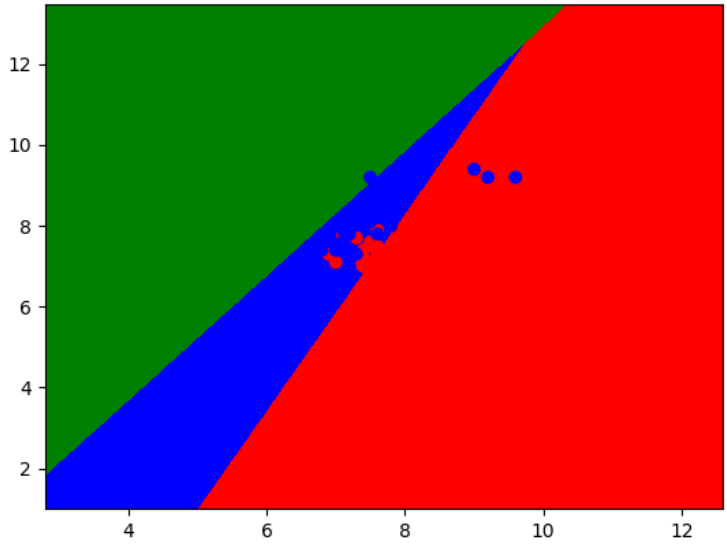

Figure 5: Separate Covariances Generative Model Plot

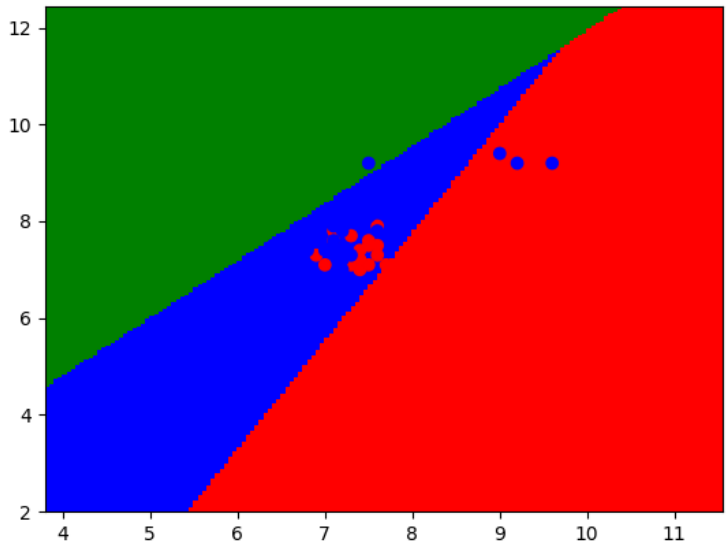

Figure 6: Logistic Regression Plot

Which classifiers model the distributions well?
Since the data is extremely difficult to classify well, given that some of the values of the fruits overlay, purely based on the appearances of the plots, I believe the Generative model with Separate Covariances did the best out of all of the 3 models.

What explains the differences?
The differences between all of the plots is that the Logistic Regression and the Shared Covariance Generative model have linear separators, whereas the Separate Covariance Generative model behaves more like multivariate Gaussians. This can be explained largely by the fact that we are incorporating all of the classification into one shared common matrix, which will limit the 'degrees of freedom' of the model, not allowing it to behave like the one with the Separate Covarience Generative models.

For logistic regression, report negative log-likelihood loss for several configurations of hyper-parameters. Why are your final choices of learning rate (η) and regularization strength (λ) reasonable? Plot loss during training for the best of these configurations, with iterations on the x-axis and loss on the y-axis (one way to do this is to add a method to the LogisticRegression Class that displays loss).

	$\eta = 0.00001$	$\eta = 0.0001$	$\eta = 0.001$
$\lambda = 0.0$	34.55	31.20	36.36
$\lambda = 0.0001$	34.56	31.20	36.38
$\lambda = 0.001$	34.58	31.26	36.64
$\lambda = 1.0$	43.48	42.25	52.26

Table 1: Table of Different Hyperparameters Configuration, reporting value of loss after 50,000 iterations

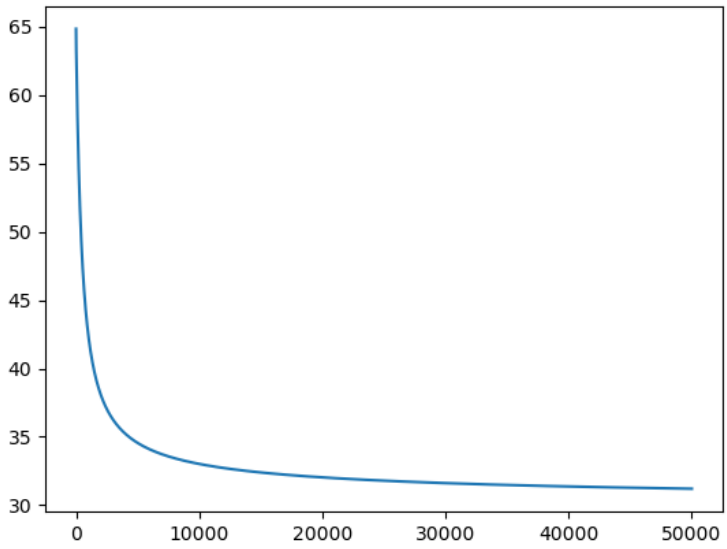

Figure 7: Logistic Regression Loss Plot for the best Hyperparameters, as highlighted in yellow in Table 1.

For both Gaussian generative models, report likelihood. In the separate covariance matrix case, be sure to use the covariance matrix that matches the true class of each data point.
Non-shared covariance likelihood is: 291.065036278
Shared covariance likelihood is: 318.265266347

End Solution

Calibration [1pt]

Approximately how long did this homework take you to complete?

Asnwer

Approx 24 hours

Problem4.py

```
# Don't change these imports. Note that the last two are the
# class implementations that you will implement in
# LogisticRegression.py and GaussianNaiveBayes.py
import matplotlib.pyplot as plt
import pandas as pd
from LogisticRegression import LogisticRegression
from GaussianGenerativeModel import GaussianGenerativeModel


## These are the hyperparameters to the classifiers. You may need to
# adjust these as you try to find the best fit for each classifier.

# Logistic Regression parameters
eta = 1E-4
lambda_parameter = 0.0001
iterations=50000

# Do not change anything below this line!!
# ----------------------------------------------------------------

# Read from file and extract X and Y
df = pd.read_csv("fruit.csv")
X = df[['width', 'height']].values
Y = (df['fruit'] - 1).values

nb1 = GaussianGenerativeModel(isSharedCovariance=False)
nb1.fit(X,Y)
nb1.visualize("generative_result_separate_covariances.png")
nb1.likelihood()

nb2 = GaussianGenerativeModel(isSharedCovariance=True)
nb2.fit(X,Y)
nb2.visualize("generative_result_shared_covariances.png")
nb2.likelihood()

lr = LogisticRegression(eta=eta, lambda_parameter=lambda_parameter, iterations=iterations)
lr.fit(X,Y)
lr.visualize('logistic_regression_result.png')
```

Listing 1: Problem 4.py Main File

GaussianGenerativeModel.py

```
from scipy.stats import multivariate_normal
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as c
import math
import pandas as pd

# Please implement the fit and predict methods of this class. You can add additional private
    methods
# by beginning them with two underscores. It may look like the __dummyPrivateMethod below.
# You can feel free to change any of the class attributes, as long as you do not change any of
# the given function headers (they must take and return the same arguments), and as long as you
# don't change anything in the .visualize() method.
class GaussianGenerativeModel:
```

```
    def __init__(self, isSharedCovariance=False):
        self.isSharedCovariance = isSharedCovariance
        self.numberOfClasses=3
        self.iterations=100

    # Just to show how to make 'private' methods
    def __dummyPrivateMethod(self, input):
        return None

    # TODO: Implement this method!
    def fit(self, X, Y):
        self.X = X
        self.Y = Y
        isSharedCovariance=self.isSharedCovariance
        numberOfClasses = self.numberOfClasses
        iterations=self.iterations

        mu=np.zeros((2,3))
        pik=[];sigma=[];valuesAll=[]

        if isSharedCovariance: sigma=np.zeros((2,2))
        for k in xrange(numberOfClasses): # looping over class K
            values=X[Y==k,:]
            valuesAll.append(values)
            YY=Y[Y==k]

            pik.append(float(len(YY))/float(len(Y)))
            mu[0,k]=np.mean([values[:,0]])
            mu[1,k] = np.mean([values[:,1]])
            if not isSharedCovariance:
                sigma.append(np.cov(values,rowvar=0))
            if isSharedCovariance:
                sigma = sigma+(float(len(values))/float(len(X)))*(np.cov(values, rowvar=0))


        self.mu=mu # pass back the values of the mean
        self.sigma=sigma # pass back the values of the variance
        self.pik=pik
        return

    def likelihood(self):
        mu=self.mu
        sigma=self.sigma
        Y=self.Y
        X=self.X
        C = np.array(pd.get_dummies(Y))
        pik=self.pik
        isSharedCovariance=self.isSharedCovariance

        lkh=0

        for k in xrange(C.shape[1]):
            for i in xrange(C.shape[0]):

                mult=np.asmatrix(X[i,:]-mu[:,k])
                if isSharedCovariance:
                    sigmaInv = np.linalg.inv(sigma)
                    lkh += (np.log(2.*math.pi)+(1./2.)*np.log(np.linalg.det(sigma))+(1./2.)*np.
    matmul(np.matmul(mult,sigmaInv),mult.T))*C[i,k]-np.log(pik[k])
                else:
                    sigmaInv = np.linalg.inv(sigma[k])
                    lkh += (np.log(2. * math.pi) + (1. / 2.) * np.log(np.linalg.det(sigma[k])) +
    (1. / 2.) * np.matmul(
                        np.matmul(mult, sigmaInv), mult.T)) * C[i, k] - np.log(pik[k])
        if isSharedCovariance:
```

```
            print 'shared covariance likelihood is: '+str(lkh[0,0])+'\n'
        else:
            print 'non-shared covariance likelihood is: '+str(lkh[0,0])+'\n'


    # TODO: Implement this method!
    def predict(self, X_to_predict):
        # The code in this method should be removed and replaced! We included it just so that the distribution code
        # is runnable and produces a (currently meaningless) visualization.
        isSharedCovariance=self.isSharedCovariance
        mu=self.mu
        pik=self.pik
        sigma=self.sigma
        predict=np.zeros((X_to_predict.shape[0],3))
        for k in xrange(3):
            if not isSharedCovariance:
                mv=multivariate_normal(mu[:,k],sigma[k]) #create a multivariate distribution for each class with shared variance
            if isSharedCovariance:
                mv = multivariate_normal(mu[:, k], sigma)
            predict[:,k]=mv.pdf(X_to_predict)*pik[k]# obtain the PDF for a given class for each of the vales of X
        Y=np.argmax(predict,axis=1)

        return np.array(Y)

    # Do not modify this method!
    def visualize(self, output_file, width=3, show_charts=False):
        X = self.X

        # Create a grid of points
        x_min, x_max = min(X[:, 0] - width), max(X[:, 0] + width)
        y_min, y_max = min(X[:, 1] - width), max(X[:, 1] + width)
        xx,yy = np.meshgrid(np.arange(x_min, x_max, .005), np.arange(y_min,
            y_max, .005))

        # Flatten the grid so the values match spec for self.predict
        xx_flat = xx.flatten()
        yy_flat = yy.flatten()
        X_topredict = np.vstack((xx_flat,yy_flat)).T

        # Get the class predictions
        Y_hat = self.predict(X_topredict)
        Y_hat = Y_hat.reshape((xx.shape[0], xx.shape[1]))

        cMap = c.ListedColormap(['r','b','g'])

        # Visualize them.
        plt.figure()
        plt.pcolormesh(xx,yy,Y_hat, cmap=cMap)
        plt.scatter(X[:, 0], X[:, 1], c=self.Y, cmap=cMap)
        plt.savefig(output_file)
        if show_charts:
            plt.show()
```

Listing 2: Gaussian Generative Model File

LogisticRegression.py

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as c
import pandas as pd
from scipy.misc import logsumexp

# Please implement the fit and predict methods of this class. You can add additional private methods
# by beginning them with two underscores. It may look like the __dummyPrivateMethod below.
# You can feel free to change any of the class attributes, as long as you do not change any of
# the given function headers (they must take and return the same arguments), and as long as you
# don't change anything in the .visualize() method.
class LogisticRegression:
    def __init__(self, eta, lambda_parameter,iterations):
        self.eta = eta
        self.lambda_parameter = lambda_parameter
        self.numberOfClasses=3
        self.iterations=iterations

    # Just to show how to make 'private' methods
    def __dummyPrivateMethod(self, input):
        return None


    # TODO: Implement this method!
    def fit(self, X, C):
        self.X = X
        self.C = C

        lambda_parameter=self.lambda_parameter
        eta = self.eta
        iterations=self.iterations
        numberOfClasses=self.numberOfClasses

        C=np.array(pd.get_dummies(C)) # convert all of the values to one-hot vectors

        Bias=np.ones((C.shape[0])) #create a bias column
        X=np.column_stack((Bias,X)) # stack the bias column

        w=np.ones((X.shape[1],numberOfClasses)) # initialize a weight matrix
        loss=[];itAll=[]
        for it in xrange(iterations):
            wx=np.matmul(X,np.transpose(w))   ### double check this to make sure its correct
            sf= np.exp(wx)  # convert the wx to the exponential form for the softmax function
            for i in xrange(wx.shape[0]): # for all the classes
                sumSf=np.sum(sf[i,:]) # create a sum vector for the soft max
                sf[i,:]=sf[i,:]/sumSf # divide each class by the sum vector for that respective class

            L=np.matmul(np.transpose((sf-C)),X)
            loss.append(-np.sum((C*np.log(sf)))+lambda_parameter*np.sum(w*w))

            w=w-eta*(L+2*lambda_parameter*w)
            itAll.append(it)
        plt.figure()
        plt.plot(itAll,loss)
        plt.savefig('logistic_regression_loss_new.png')

        print 'the final loss for the logistic regression model is: '+str(loss[-1])+'\n'

        self.weights=w
        return

    # TODO: Implement this method!
    def predict(self, X_to_predict):
```

```
        # The code in this method should be removed and replaced! We included it just so that the distribution code
        # is runnable and produces a (currently meaningless) visualization.
        w=self.weights

        Bias = np.ones((X_to_predict.shape[0]))  # create a bias column
        X_to_predict = np.column_stack((Bias, X_to_predict))  # stack the bias column

        Y=np.argmax(np.matmul(X_to_predict,np.transpose(w)),axis=1)

        return Y

    def visualize(self, output_file, width=2, show_charts=False):
        X = self.X

        # Create a grid of points
        x_min, x_max = min(X[:, 0] - width), max(X[:, 0] + width)
        y_min, y_max = min(X[:, 1] - width), max(X[:, 1] + width)
        xx,yy = np.meshgrid(np.arange(x_min, x_max, .05), np.arange(y_min,
            y_max, .05))

        # Flatten the grid so the values match spec for self.predict
        xx_flat = xx.flatten()
        yy_flat = yy.flatten()
        X_topredict = np.vstack((xx_flat,yy_flat)).T

        # Get the class predictions
        Y_hat = self.predict(X_topredict)
        Y_hat = Y_hat.reshape((xx.shape[0], xx.shape[1]))

        cMap = c.ListedColormap(['r','b','g'])

        # Visualize them.
        plt.figure()
        plt.pcolormesh(xx,yy,Y_hat, cmap=cMap)
        plt.scatter(X[:, 0], X[:, 1], c=self.C, cmap=cMap, lw=1)
        plt.savefig(output_file)
        if show_charts:
            plt.show()
```

Listing 3: Logistic Regression Model File