Matthew Leifer
matthewleifer@college.harvard.edu
CS181-S17

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

Collaborators: John Doe, Fred Doe

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 $\mathbf{w}$ 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

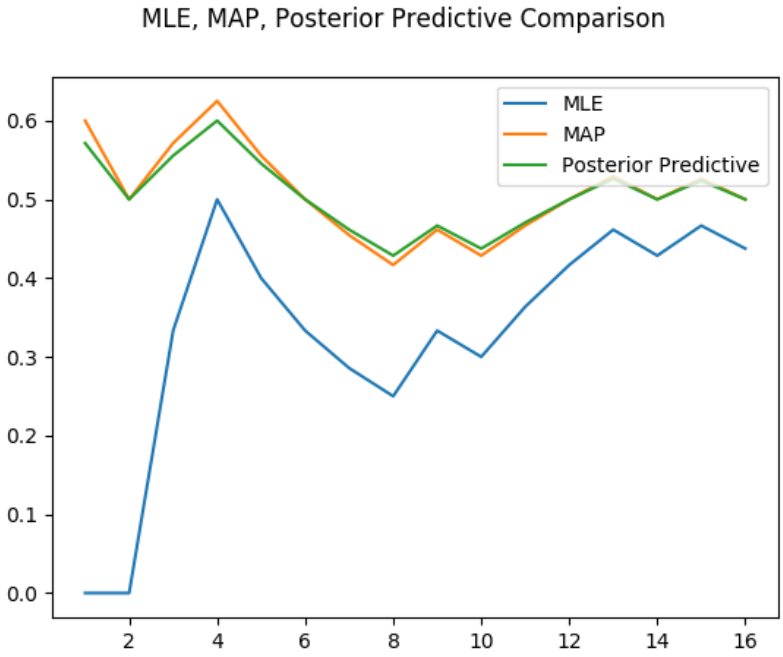
$$\mathbf{S}_n = (\alpha\mathbf{I} + \beta\mathbf{X}^\top\mathbf{X})^{-1}$$
$$\mathbf{m}_n = \beta\mathbf{S}_n\mathbf{X}^\top\mathbf{y}$$

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?

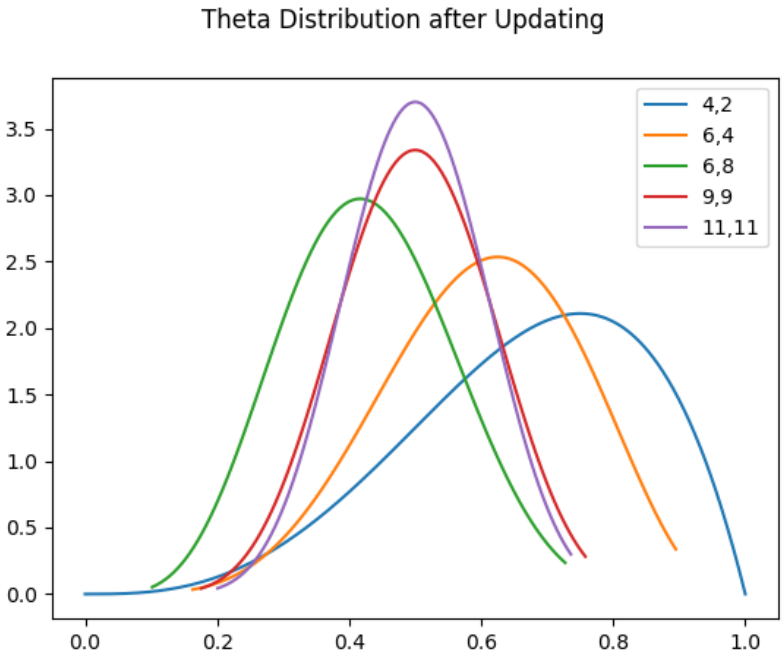
Solution

1. Let n_0 be the number of 0's and let n_1 be the number of 1's seen. $\Theta \sim \text{Beta}(4, 2)$

$$\Theta_{MLE} = \frac{n_1}{n_0 + n_1} \qquad \Theta_{MAP} = \frac{\alpha + n_1 - 1}{\alpha + \beta + n_0 + n_1 - 2} \qquad \Theta_{Post.pred.} = \frac{\alpha + n_1}{\alpha + \beta + n_0 + n_1}$$



2. .



3. The MAP and posterior predictive distributions are pretty close throughout the simulation but the MLE differs significantly because it makes no use of the prior $\theta \sim$Beta(4,2) and only 'looks' at the data that has already come in. MAP and the posterior are influenced by our choice of a prior.

4. We would want to use an isotropic prior on the weights if we have no prior knowledge of what the weights should be - it's a good prior to use when we know nothing.

5. We want to figure out what $\mathbf{w}$ is after we've seen data D, i.e. what is $p(\mathbf{w}|D)$.
   By Bayes rule, $p(w|D) = \frac{p(\mathbf{w})p(D|\mathbf{w})}{P(D)}$ and so, taking logs we see that
   $\log p(\mathbf{w}|D) = c_1 + \log p(\mathbf{w}) + \log p(D|\mathbf{w})$
   $= c_1 - \frac{1}{2}[\mathbf{w}^T\alpha\mathbf{w} + (\mathbf{y} - \mathbf{X}\mathbf{w})^T\beta(\mathbf{y} - \mathbf{X}\mathbf{w})]$
   Now completing the square and pushing extra terms into c_1
   $= c_1 - \frac{1}{2}[\mathbf{w}^T\alpha\mathbf{w} - 2\mathbf{w}^T\alpha\mathbf{0} - 2\mathbf{w}^T\mathbf{X}^T\beta\mathbf{y} + (\mathbf{X}\mathbf{w})^T\beta(\mathbf{X}\mathbf{w})]$
   $= c_1 - \frac{1}{2}[\mathbf{w}^T(\alpha\mathbf{I} + \beta\mathbf{X}^T\mathbf{X})\mathbf{w} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{y}]$
   From which we can see that, $S_n = (\alpha\mathbf{I} + \beta\mathbf{X}^T\mathbf{X})^{-1}$ and $m_n = \beta S_n\mathbf{X}^T\mathbf{y}$ which is what we wanted to show.

6. Θ_{MAP} is $m_n = \beta S_n\mathbf{X}^T\mathbf{y}$. So if we have a weak prior, $\alpha << 1$ and so $S_n = (\alpha\mathbf{I} + \beta\mathbf{X}^T\mathbf{X})^{-1} \rightarrow \beta^-1(\mathbf{X}^T\mathbf{X})$. Therefore $m_n = \beta\beta^{-1}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}\mathbf{y} = \Theta_{MLE}$

   So if we have a weak prior, ΘMAP approaches Θ_{MLE}.

7. When we use a prior that is neither weak nor strong, we recover the solution to ridge regression.

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

1. The kth component of the softmax function on vector $\mathbf{z}$ is $\frac{\exp(z_k)}{\sum_{l=1}^{c}\exp(z_l)}$ which must be less than one becuase $\exp(x)$ for all real x is greater than 0 and so the numerator of this fraction cannot be larger than the denominator and so each component in the output vector is at most 1.

2. $\sum_{k=1}^{c}\frac{\exp(z_k)}{\sum_{l=1}^{c}\exp(z_l)} = \frac{\sum_{k=1}^{c}\exp(z_k)}{\sum_{l=1}^{c}\exp(z_l)}$, which clearly equals 1.

3. By definition $\text{softmax}(\mathbf{z})_k = \frac{\exp(z_k)}{\sum_{i=1}^{c}\exp(z_i)}$ and $\text{softmax}(\mathbf{z})_l = \frac{\exp(z_l)}{\sum_{i=1}^{c}\exp(z_i)}$
   Let $z_k < z_l$
   $\exp(z_k) < \exp(z_l)$ because the exponential function is strictly increasing
   $\frac{\exp(z_k)}{\sum_{i=1}^{c}\exp(z_i)} < \frac{\exp(z_l)}{\sum_{i=1}^{c}\exp(z_i)}$ and so the softmax function preserves order.

4. For $j \neq k$, $\frac{d}{dz_j}\text{softmax}(\mathbf{z})_k$
   $= \frac{d}{dz_j}\frac{e^{z_k}}{e^{z_1}+...+e^{z_c}}$
   $= \frac{0\cdot(\sum_{i=1}^{c}e^{z_i}) - e^{z_k}(\sum_{i=1}^{c}e^{z_i})}{(\sum_{i=1}^{c}e^{z_i})^2}$
   $= \frac{e^{z_k}}{(\sum_{i=1}^{c}e^{z_i})}(-\frac{e^{z_j}}{(\sum_{i=1}^{c}e^{z_i})})$
   $= \text{softmax}(\mathbf{z})_k(-\text{softmax}(\mathbf{z})_j)$

   For $j = k$, $\frac{d}{dz_k}\text{softmax}(\mathbf{z})_k$
   $= \frac{e^{z_k}(\sum_{i=1}^{c}e^{z_i}) - e^{z_k}e^{z_k}}{(\sum_{i=1}^{c}e^{z_i})^2}$
   $= \frac{e^{z_k}}{\sum_{i=1}^{c}e^{z_i}}(1-\frac{e^{z_k}}{\sum_{i=1}^{c}e^{z_i}})$
   $= \text{softmax}(\mathbf{z})_k(1-\text{softmax}(\mathbf{z})_k)$

5. $\frac{\partial}{\partial\mathbf{w}_k}\mathcal{L}(\{\mathbf{w}_\ell\}) = \frac{\partial}{\partial\mathbf{w}_k}\big(-\sum_{i=1}^{n}\log(\text{softmax}([\mathbf{w}_1^T x_i ... \mathbf{w}_c^T x_i]^T))\big)$
   $= -\sum_{i=1}^{n}\frac{1}{\text{softmax}([\mathbf{w}_1^T x_i ... \mathbf{w}_c^T x_i]^T)}\cdot\frac{\partial}{\partial\mathbf{w}_k}\text{softmax}([\mathbf{w}_1^T x_i ... \mathbf{w}_c^T x_i]^T)$
   $= -\sum_{i=1}^{n}\frac{1}{\text{softmax}([\mathbf{w}_1^T x_i ... \mathbf{w}_c^T x_i]^T)}\text{softmax}([\mathbf{w}_1^T x_i ... \mathbf{w}_c^T x_i]^T)(y_{ik} - \text{softmax}([\mathbf{w}_1^T x_i ... \mathbf{w}_c^T x_i]^T))\mathbf{x}_i$
   $= \sum_{i=1}^{n}(y_{ik} - \text{softmax}([\mathbf{w}_1^T x_i ... \mathbf{w}_c^T x_i]^T))\mathbf{x}_i$

Problem 3 (Return of matrix calculus, 10pts)

Consider now a generative c-class model. We adopt class prior $p(\mathbf{y} = C_k; \pi) = \pi_k$ 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, $-\ln p(D; \pi)$.
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

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

and different means μ_k for each class.

3. Derive the gradient of the negative 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. Derive the maximum-likelihood estimator for vector μ_k. Once again, your final answer should seem intuitive.
5. Derive the gradient for the negative log-likelihood with respect to the covariance matrix $\mathbf{\Sigma}$ (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

1. $p(D;\pi)\alpha\prod_{i=1}^{n} p(\mathbf{x}_i;\pi) = \prod_{i=1}^{n}\prod_{k=1}^{c}(\pi_k p(\mathbf{x}_i|y_i = C_k))^{y_{ik}}$

   (I dropped the denominator which doesn't depend on π_k) $-\ln p(D;\pi) = -\sum_{i=1}^{n}\sum_{k=1}^{c} y_{ik}\ln\pi_k + y_{ik}\ln p(\mathbf{x}_i|y_i = C_k)$

2. $L(\pi,\lambda) = -\sum_{i=1}^{n}\sum_{k=1}^{c} y_{ik}\ln\pi_k + y_{ik}\ln p(\mathbf{x}_i|y_i = C_k) + \lambda(\sum_{k=1}^{c}\pi_k - 1)$

   $\frac{\partial}{\partial\pi_j}L(\pi,\lambda) = -\sum_{i=1}^{n}\frac{y_{ij}}{\pi_j} - \lambda = 0$

   $\frac{\partial}{\partial\lambda}L(\pi,\lambda) = \sum_{k=1}^{c}\pi_k = 1$

   So, $\sum_{k=1}^{c}\sum_{i=1}^{n}\frac{y_{ik}}{\lambda} = 1 \rightarrow \lambda = \sum_{i=1}^{n}\sum_{k=1}^{c} y_{ik}$.

   This means that for all j, $\hat{\pi_j} = \frac{\sum_{i=1}^{n} y_{ij}}{\sum_{i=1}^{n}\sum_{k=1}^{c} y_{ik}}$, which intuitively makes sense because this fraction equals the number of things in class j divided by the total number of things so that's why the MLE is this.

3. $-\ln p(D) = -\sum_{i=1}^{n}\sum_{k=1}^{c} y_{ik}\ln(\pi_k) + y_{ik}(-\frac{1}{2}(\mathbf{x}_i - \mu_k)^T\Sigma^{-1}(\mathbf{x}_i - \mu_k) + c_1$

   Now taking the gradient and droping the terms without μ_k

   $\frac{\partial}{\partial\mu_k} - \ln p(D) = \frac{\partial}{\partial\mu_k} - \sum_{i=1}^{n}\sum_{k=1}^{c}\frac{y_{ik}}{2}(\mathbf{x}_i - \mu_k)^T\Sigma^{-1}(\mathbf{x}_i - \mu_k)$

   $= -\sum_{i=1}^{n}\frac{y_{ik}}{2}(-2\Sigma^{-1}\mathbf{x}_i + 2\Sigma^{-1}\mu_k)$

   $= \sum_{i=1}^{n} y_{ik}(\mathbf{x}_i - \mu_k)\Sigma^{-1}$

4. $\sum_{i=1}^{n} y_{ik}(\mathbf{x}_i - \mu_k)\Sigma^{-1} = 0$

   $\sum_{i=1}^{n} y_{ik}x_i = \mu_k\sum_{i=1}^{n} y_{ik}$

   $\mu_k = \frac{1}{\sum_{i=1}^{n} y_{ik}}\sum_{i=1}^{n} y_{ik}x_i$ Let N_k be $\sum_{i=1}^{n} y_{ik}$, i.e. the number of samples that class k. Therefore, $\mu_k = \frac{1}{N_k}\sum_{i=1}^{n} y_{ik}x_i$. So μ_k is the average of all the $\mathbf{x}$'s that were classified into group k.

5. $\frac{\partial}{\partial\Sigma} - \ln p(D) = \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{c} y_{ik}\Sigma^{-T} - \Sigma^{-T}(\mathbf{x}_i - \mu_\mathbf{k})(\mathbf{x}_i - \mu_\mathbf{k})^T\Sigma^{-T}$

Note that $\Sigma^{-T} = \Sigma^{-1}$

$\frac{n}{2}\Sigma^{-1} - \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{c} y_{ik}\Sigma^{-1}(\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^T\Sigma^{-1}$

6. $\frac{n}{2}\Sigma^{-1} - \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{c} y_{ik}\Sigma^{-1}(\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^T\Sigma^{-1} = 0$

   $n = \sum_{i=1}^{n}\sum_{k=1}^{c} y_{ik}(\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^T\Sigma^{-1}$

   $\Sigma = \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{c} y_{ik}(\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^T$

   Let $S_k = \sum_{y_i \in C_k} (\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^T = N_k(\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^T$ where $N_k = \sum_{i=1}^{n} y_{ik}$

   So, $\Sigma = \frac{1}{n}(S_1 + S_2 + ... + S_k)$

4. 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
...
```

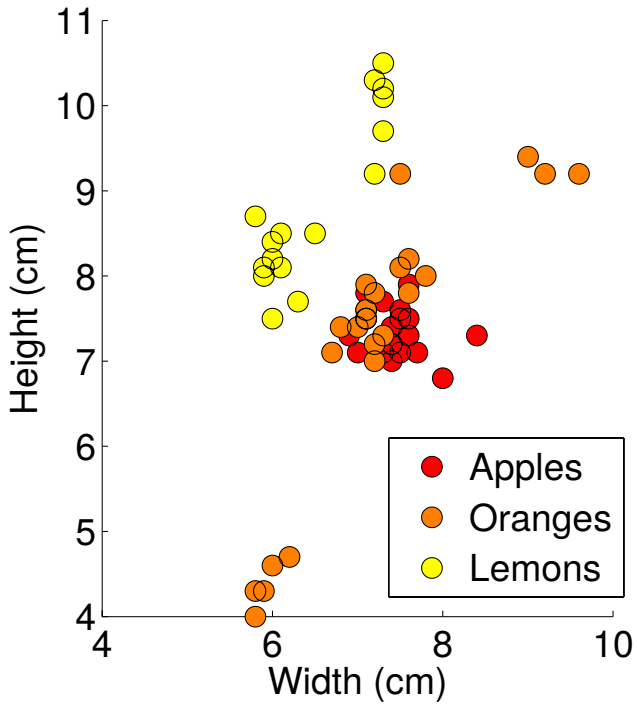


Figure 1: Heights and widths of apples, oranges, and lemons. These fruit were purchased and measured by Iain Murray: http://homepages.inf.ed.ac.uk/imurray2/teaching/oranges_and_lemons/.

Problem 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`.

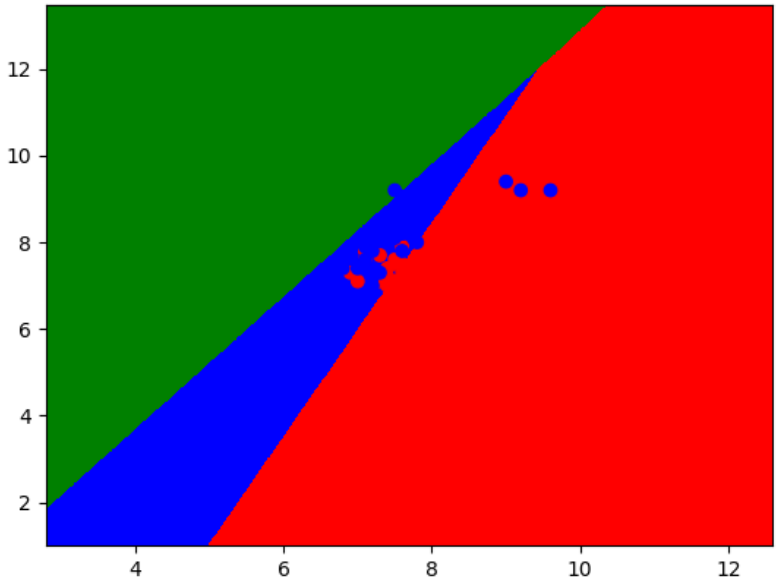
You will be given class interfaces for `GaussianGenerativeModel` and `LogisticRegression` in the distribution code, and the code will indicate certain lines that you should not change in your final submission. Naturally, don't change these. These classes will allow the final submissions to have consistency. 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.

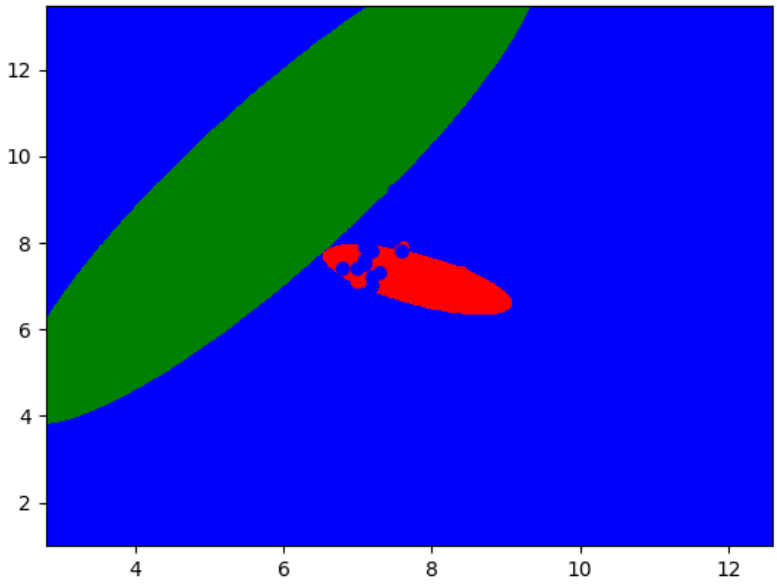
- The `visualize()` method for each classifier will save a plot that will show the decision boundaries. You should include these in this assignment.
- Which classifiers model the distributions well?
- What explains the differences?

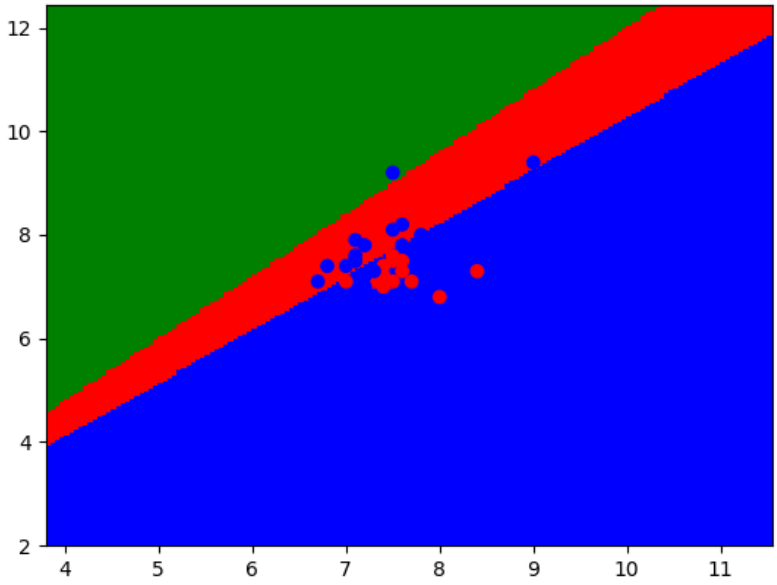
In addition to comparing the decision boundaries of the three models visually:

- 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.

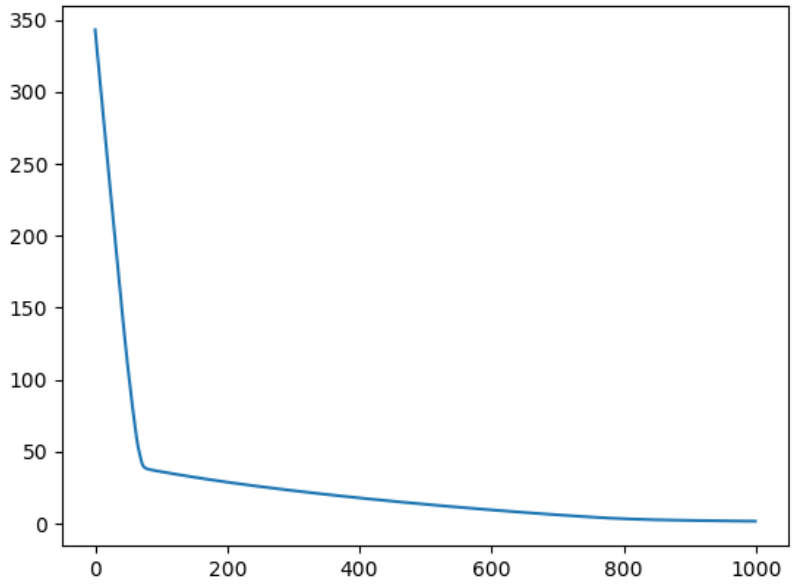
1. The first graph is the generative gaussian classifier with shared covariances. It had a negative log likelihood loss of 777.8. The next graph is the gaussian classifier with separate covariances. It had a log likelihood loss of 158.06. The last graph is the logistic regression classifier which after 1000 interations and using $\eta = 0.001$ and $\lambda = 0.001$ had a likelihood loss of 0.758, included below is a table of η, λ and the loss of the model for those values. These rates seem reasonable because they need to be small so that the gradient descent will converge. The shared covariances led to the linear boundaries whereas the classifier that used separate covariance matrices had quadratic (in this case elliptical) boundaries. The classifier that misclassified the fewest points was the Gaussian separate covariance classifier.


12
10
8
6
4
2
4
6
8
10
12


12
10
8
6
4
2
4
6
8
10
12



Lambda = 0.001; Eta = 0.001



η	λ	Loss
0.01	0.01	10.0291806236
0.001	0.01	0.953390449527
0.0001	0.01	1.00744303184
1e-05	0.01	1.08027622679
0.01	0.001	6.76722525374
0.001	0.001	0.758400464784
0.0001	0.001	0.772650945619
1e-05	0.001	1.06869419021
0.01	0.0001	5.75862920411
0.001	0.0001	0.94039775188
0.0001	0.0001	11.0042596124
1e-05	0.0001	35.9436282172
0.01	1e-05	1.20499926242
0.001	1e-05	0.75924445648
0.0001	1e-05	19.9209830895
1e-05	1e-05	103.245749749

Calibration [1pt]

Approximately how long did this homework take you to complete? Easily more than 20 hours. The only real problem I had was with the Logistic Regression because the multiclass took ages to figure out and then so did getting the right parameters.