

CPSC 440/540 Machine Learning (January-April, 2022)

Assignment 1 (due Friday January 21st at midnight)

IMPORTANT!!!! Please carefully read the submission instructions that will be posted on Piazza. We will deduct 50% on assignments that do not follow the instructions.

Most of the questions below are related to topics covered in CPSC 340, or other courses listed on the prerequisite form. There are several “notes” available on the webpage which can help with some relevant background.

If you find this assignment to be difficult overall, that is an early warning sign that you may not be prepared to take CPSC 440 at this time. Future assignments may be longer and more difficult than this one.

We use blue to highlight the deliverables that you must answer/do/submit with the assignment.

Basic Information

1. Name:

Answer: Masoud Mokhtari

2. Student ID:

Answer: 14186167

1 Very-Short Answer Questions

Give a short and concise 1-2 sentence answer to the below questions. All acronyms, symbols, and methods are as used in CPSC 340.

Answer: *Citation: To answer the following questions, I have used CPSC340 notes from previous offerings.*

1. Suppose that a famous machine learning personality is advertising their “extremely-deep convolutional fuzzy-genetic Hilbert-long-short adversarial-recurrent neural network” classifier, which has 500 hyperparameters. This person claims that if you take 10 different famous datasets, and tune the 500 hyperparameters based on each dataset’s validation set, that you can beat the current best-known validation set error on all 10 datasets. Explain whether or not this amazing claim is likely to be meaningful.

Answer: This is not a meaningful achievement since finding the optimal set over these high number of hyperparameters requires a large number of trials on the validation sets. **This causes optimization bias to appear which means that our validation sets are a biased estimator of test sets.** This somehow breaks the **golden rule of ML** by treating the validation set as a training set.

2. Consider fitting a neural network with one hidden layer. Would we call this a parametric or a non-parametric model if the hidden layer had n units (where n is the number of training examples)? Would it be parametric or non-parametric if it had d^2 units (where d is the number of input features)?

Answer: In the parametric case, size of the model is $O(1)$ with respect to n , whereas in the non-parametric case, size of the model depends on n . Therefore, **the first case is non-parametric and the second case is parametric.**

3. Suppose we want to solve the binary classification problem of determining whether 28 by 28 images were images of digits or images of letters. What is an easy way to make our classifier invariant to small translations of the training images?

Answer: One way to add invariance is to **add translated images to the training set** (data augmentation).

4. Ensemble methods are models that combine the predictions of multiple individual models. Give a reason why an ensemble method could do better than the best individual model in the ensemble.

Answer: **If individual models overfit in different ways, ensemble methods such as averaging can reduce the effect of each individual model’s error.** For example, if one of the models make a wrong prediction on a sample while the majority of models make the right prediction, the overall ensemble model would choose the correct prediction.

5. A common method for outlier detection is to collect a large number of outliers, and build a binary classifier that distinguishes these outliers from standard “inlier” datapoints. What is an advantage and a drawback of this approach to outlier detection, compared to unsupervised outlier detection methods?

Answer: For supervised outlier detection (binary classifier), **we need prior knowledge of what outliers look like and we may not be able to find new types of outliers**, whereas unsupervised methods do not require such knowledge in the form of labels. The advantage of such an approach is that we can use **supervised methods to find complicated patterns of outliers.**

6. Why do we typically add a column of 1 values to X when we do linear regression?

Answer: **This trick is used to accommodate for the bias term in a seamless manner by incorporating it into the weights matrices, which prevents the model from being forced to go through the origin.**

7. If a function is convex, what does that say about stationary points of the function? Does convexity imply that a stationary points exists?

Answer: For a convex function, **all stationary points correspond to the same global minimum of the function. Convexity does not necessarily mean that a stationary point exists ($|x|$ is one example).**

8. What is the cost in $O()$ notation for fitting a linear regression model with n examples and 1 feature using least squares? Explain under what conditions it would make sense to use gradient descent instead of the normal equations for fitting a linear regression model with 1 feature.

Answer: With $d = 1$, we have **$O(n)$ and $O(n^2)$ for normal equation and gradient descent methods respectively.** Normal equations only work for least squares problems. Therefore, **if we had other interesting loss types and problems where a closed analytic formula may not be possible or easy to come up with, gradient descent would be the correct method to use.**

9. Suppose you are hired by company to look at a dataset, and they want to “find which features are relevant” for their prediction task. What are 2 warnings you would give them regarding the features found by feature selection methods?

Answer: **(1)Some features may seem relevant because some information is missing (Taco Tuesday example). Confounding issue may cause irrelevant information to appear relevant. (2) Relevant information may actually have very small effects in correct predictions.**

10. If we fit a linear regression model with L1-regularization of the parameters, what is the effect of the regularization parameter λ on the sparsity level of the solution? What is the effect of λ on the two parts of the fundamental trade-off (the training error and the amount of overfitting)?

Answer: **Higher λ values with L1-regularization encourages sparser weights by pushing values towards zero. Higher λ values with L1-regularization increases training error and reduces amount of overfitting.**

11. Suppose we fit a one-feature linear regression model with a polynomial basis, and this leads to non-zero regression weights. What would the prediction of this model be as the feature value x^i goes to ∞ ? How would this change if you used Gaussian RBFs as features?

Answer: **In the polynomial basis case, the prediction would also be $+$ or $-$ infinity. In the Gaussian RBF case because local bumps are used instead of global polynomials, based on weights, a finite prediction would be produced and if x is far away from training samples, the prediction is 0.**

12. When searching for a good w for a linear classifier, why do we use the logistic loss instead of just minimizing the number of classification errors?

Answer: **0-1 or classification error loss is non-convex in w and gradient is zero everywhere, making it a hard problem for optimization. On the other hand, logistic loss does not produce degenerate results, is differentiable and convex, which means it can be optimized with gradient descent.**

13. With stochastic gradient descent, the loss might go up or down each time the parameters are updated. However, we don't actually know which of these cases occurred. Explain why it doesn't make sense to check whether the loss went up/down after each update.

Answer: **While individual gradient updates may push the model in the wrong direction, it's the average over all updates that would eventually push the model in the right direction.**

14. Suppose we are given two vectors of integers, $x=(x_1, x_2, \dots, x_n)$ and $y=(y_1, y_2, \dots, y_n)$, and we want to find the longest sequence of numbers that appears consecutively in both x and y . Describe an $O(n^2)$ algorithm to do this using dynamic programming (you can assume the numbers have a fixed-sized representation).

Answer: We can build an $n \times n$ table and fill it with longest common sequence for each sub-sequence using dynamic programming. The largest number in the table would then show the longest common consecutive sequence.

Citation: to solve this question, the following references were used:

- <https://www.youtube.com/watch?v=UZRkpGk943Q>
- https://en.wikipedia.org/wiki/Longest_common_substring_problem

Data: $x=(x_1, x_2, \dots, x_n)$, $y=(y_1, y_2, \dots, y_n)$

longest_len = 0;

last_index = 0;

dp_array = array(n, n);

for i **in** $1:n$ **do**

for j **in** $1:n$ **do**

if $x[i] == y[j]$ **then**

if $i==1$ **and** $j==1$ **then**

 dp_array[i, j] = 1;

end

else

 dp_array[i, j] = 1 + dp_array[i-1, j-1];

end

if longest_len < dp_array[i, j] **then**

 longest_len = dp_array[i, j];

 last_index = i;

end

end

else

 dp_array[i, j] = 0;

end

end

end

if last_index == 0 **then**

 return None

end

else

 return $x[\text{last_index} - \text{longest_len} + 1 : \text{last_index}]$;

end

Algorithm 1: Longest Integer Sequence - Dynamic Programming

Note: This algorithm only assumes one longest sequence exists (it can be easily modified to save the last index for all sequences of maximum length).

15. Given a vector of n positive integers (x_1, x_2, \dots, x_n) , give an $O(n)$ time algorithm for computing, for each position i , the sum of all numbers except x_i . Next, describe how you could solve this problem in $O(n)$ even if you were not allowed to use subtraction/negation (hint: you can start at the beginning or the end of the list).

Answer: For the first case, we can compute the sum for all number and only subtract the value at each position for the answer at that position:

```
Data: x=( $x_1, x_2, \dots, x_n$ )
sum = 0;
output_array = array(n);
for k in 1:n do
    | sum += x[k];
end
for k in 1:n do
    | output_array[k] = sum-x[k];
end
return output_array;
```

Algorithm 2: Sum of Sequence Except one Element - with Subtraction

If we are not allowed to use subtraction, this can still be solved in $O(n)$. As mentioned in the hint, we can first traverse the list from the beginning and keep track of sum of elements before each element. We can then start from the end and keep track of sum of elements after each element. This allows us to solve the problem with two separate loops of $O(n)$. Citation: the solution was inspired by the Product of Array Except Itself problem: <https://www.geeksforgeeks.org/a-product-array-puzzle/>

```
Data: x=( $x_1, x_2, \dots, x_n$ )
sum = 0;
output_array = array(n);
for k in 1:n do
    | output_array[k] = sum;
    | sum += x[k];
end
sum = 0;
for k in n:1 do
    | output_array[k] += sum;
    | sum += x[k];
end
return output_array;
```

Algorithm 3: Sum of Sequence Except one Element - without Subtraction

16. Consider using a fully-connected neural network for 3-class classification on a problem with $d = 10$. If the network has one hidden layer of size $k = 100$, how many parameters (including biases), does the network have?

Answer: The number of parameters is computed as: $(10 * 100 + 100) + (100 * 3 + 3) = 1403$

2 Coding Questions

The questions below use code contained in *a1.zip*, which you can download from the course webpage. The scripts mentioned in the question can be run in Julia's REPL in the directory containing the extracted files. If you have not previously used Julia, there is a list of useful Julia commands (and syntax) among the list of notes on the course webpage.

Several scripts use packages. For example, the *JLD* package is used to load data and the *PyPlot* package to make plots. If you have not previously used these packages, they can be installed using:

```
using Pkg
Pkg.add("JLD")
Pkg.add("PyPlot")
```

2.1 K-Means Clustering

If you run the function *example_Kmeans*, it will load a dataset with two features and a very obvious clustering structure. It will then apply the *k*-means algorithm with a random initialization. The result of applying the algorithm will thus depend on the randomization, but a typical run might look like this:

(Note that the colours are arbitrary due to the label switching problem.) But the “correct” clustering (that was used to make the data) is something more like this:

If you run the demo several times, it will find different clusterings. To select among clusterings for a *fixed* value of *k*, one strategy is to minimize the sum of squared distances between examples x^i and their means w_{y^i} ,

$$f(w_1, w_2, \dots, w_k, y^1, y^2, \dots, y^n) = \sum_{i=1}^n \|x^i - w_{y^i}\|_2^2 = \sum_{i=1}^n \sum_{j=1}^d (x_j^i - w_{y^i j})^2.$$

where y^i is the index of the closest mean to x^i . This is a natural criterion because the steps of *k*-means alternately optimize this objective function in terms of the w_c and the y^i values.

1. Write a new function called *kMeansError* that takes in a dataset *X*, a set of cluster assignments *y*, and a set of cluster means *W*, and computes this objective function. Hand in your code.

Answer: The implementation is shown in Figure 1.

```
2 function kMeansError(X, y, W)
3     # Get the dimensions and ensure they are correct
4     (nx, dx) = size(X);
5     (ny, ) = size(y);
6     (nt, dt) = size(W);
7     @assert(nx==ny)
8     @assert(dx==dt)
9
10    # Initialize error
11    ssd = 0
12
13    # Non-vectorized solution
14    for i in 1:nx
15        for j in 1:dx
16            ssd += (X[i, j] - W[y[i], j])^2
17        end
18    end
19
20 end
21
```

Figure 1: Non-vectorized implementation for *kMeansError*

2. Instead of printing the number of labels that change on each iteration, what trend do you observe if you print the value of *kMeansError* after each iteration of the k-means algorithm?

Answer: The k-means error is reduced at higher rates for the initial iterations, and the reduction rate decreases until convergence, where no more change is observed. It must be noted that the initial and converged error depend on the initialization.

3. Using the *clustering2Dplot* file, output the clustering obtained by running k-means 50 times (with $k = 4$) and taking the one with the lowest error. Note that the k-means training function will run much faster if you set `doPlot = false` or just remove this argument.

Answer: The clusters in Figure 2 had the lowest k-means error among 50 trials (error = 3071.4681)

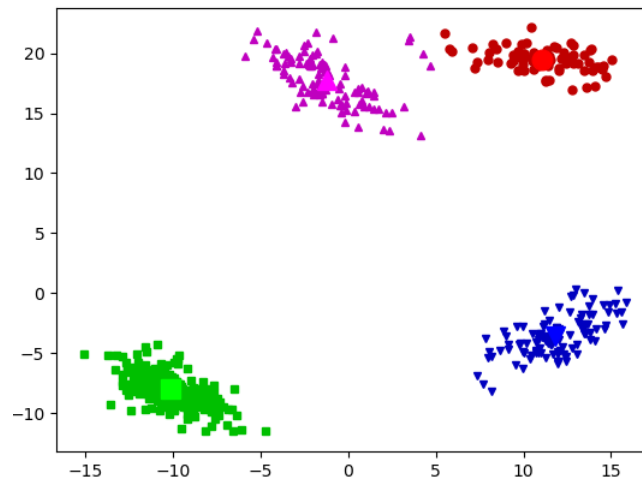


Figure 2: Clusters for the kmeans trial with lowest kMeansError.

4. Explain why the *kMeansError* function should not be used to choose k .

Answer: kMeansError is not suitable to find k since (for example) setting k to be equal to the number of data samples would cause this error to be 0, even though these clusters are not a helpful representation of the data.

5. Explain why even evaluating the *kMeansError* function on test data still wouldn't be a suitable approach to choosing k .

Answer: Again, as an example, if the number of clusters is equal to the number of samples, kMeansError would be minimized (not necessarily to 0) even for test samples. In essence, this makes the algorithm similar to 1-nearest-neighbour algorithm rather than properly associating test samples with meaningful clusters.

6. The data in *clusterData2.jld* is the exact same as *clusterData.jld*, except it has 4 outliers that are far away from the data. Using the *clustering2Dplot* function, output the clustering obtained by running k-means 50 times (with $k = 4$) on *clusterData2.jld* and taking the one with the lowest error. Are you satisfied with the result?

Answer: As shown in Figure 3, the results are not satisfactory. For one, two distinctive clusters that were properly distinguished for *clusterData* (purple and red in Figure 2) are now in the same cluster. Additionally, one of the outliers is now a cluster of its own, which is not informative. In conclusion, our current k-means algorithm is very sensitive to outliers.

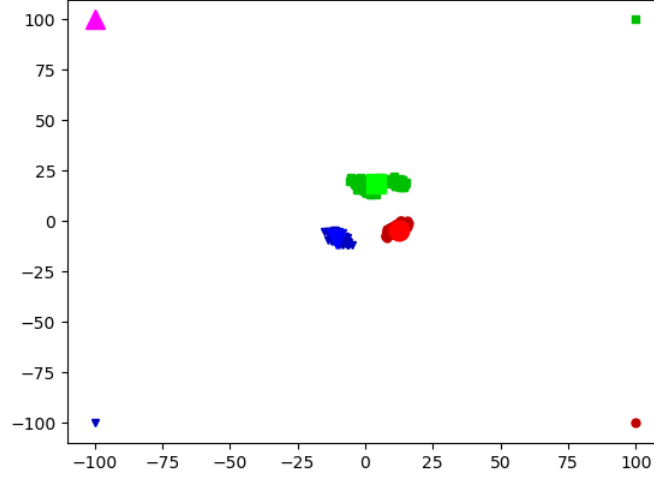


Figure 3: Clusters for the clusterData2.

7. Implement the *k-medians* algorithm, which assigns examples to the nearest w_c in the L1-norm and then updates the w_c by setting them to the “median” of the points assigned to the cluster (we define the d -dimensional median as the concatenation of the medians along each dimension). For this algorithm it makes sense to use the L1-norm version of the error (where y_i now represents the closest median in the L1-norm),

$$f(w_1, w_2, \dots, w_k, y_1, y_2, \dots, y_n) = \sum_{i=1}^n \|x_i - w_{y_i}\|_1 = \sum_{i=1}^n \sum_{j=1}^d |x_{ij} - w_{y_i j}|,$$

Hand in your code and plot obtained by taking the clustering with the lowest L1-norm after using 50 random initializations for $k = 4$.

Answer: As shown in Figure 4, using L1 norm, the model is a lot more robust to outliers. The same code given for kMeans was used but square distance and mean were replaced by l1Distance and median respectively as shown in Figure 7. The implementation for L1 error and distance are shown in Figures 5 and 6.

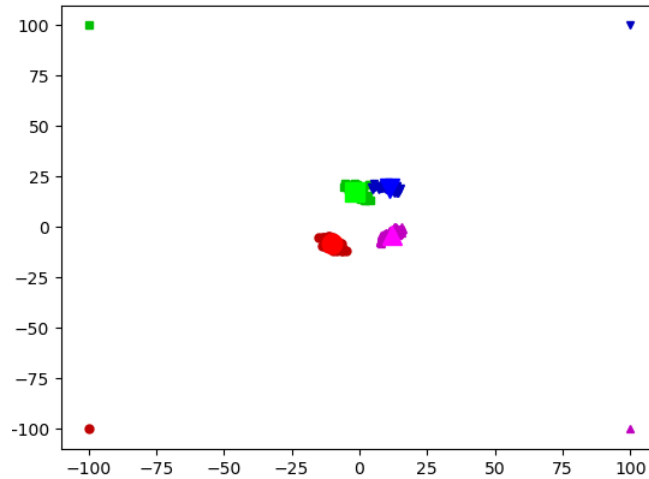


Figure 4: Clusters for the clusterData2 using L1 Norm with K-medians.

```

26
27 # Return L1 distance of all pairs of rows in X1 and X2
28 using LinearAlgebra
29 function l1Distance(X1,X2)
30     (n,d) = size(X1)
31     (t,d2) = size(X2)
32     @assert(d==d2)
33
34     # Create output array
35     D = zeros(n, t);
36     for i in 1:n
37         for j in 1:t
38             D[i, j] = norm(X1[i, :] .- X2[j, :], 1);
39         end
40     end
41     D;
42 end
43

```

Figure 5: Julia implementation for L1 Distance used by the K-Medians algorithm.

```

1 using LinearAlgebra
2 function l1KMediansError(X, y, W)
3     # Get the dimensions and ensure they are correct
4     (nx, dx) = size(X);
5     (ny, ) = size(y);
6     (nt, dt) = size(W);
7     @assert(nx==ny)
8     @assert(dx==dt)
9
10    # Initialize error
11    l1error = 0;
12
13    # Non-vectorized solution
14    for i in 1:nx
15        for j in 1:dx
16            l1error += norm(X[i, j] - W[y[i], j], 1);
17        end
18    end
19
20    l1error;
21
22 end

```

Figure 6: L1 error implementation.

```

13
14 function l1KMedians(X,k;doPlot=false)
15     # K-medians clustering with L1 Norm
16
17     (n,d) = size(X)
18
19     # Choose random points to initialize medians
20     W = zeros(k,d)
21     perm = randperm(n)
22     for c = 1:k
23         W[c,:] = X[perm[c],:]
24     end
25
26     # Initialize cluster assignment vector
27     y = zeros{Int64, n}
28     changes = n
29     while changes != 0
30
31         # Compute L1 distance between each point and each median
32         D = l1Distance(X,W)
33
34         # Degenerate clusters will distance NaN, change to Inf
35         # (since Julia thinks NaN is smaller than all other numbers)
36         D[findall(isnan.(D))] .= Inf
37
38         # Assign each data point to closest median (track number of changes labels)
39         changes = 0
40         for i in 1:n
41             (~,y_new) = findmin(D[i,:])
42             changes += (y_new != y[i])
43             y[i] = y_new
44         end
45
46         # Optionally visualize the algorithm steps
47         if doPlot && d == 2
48             clustering2Dplot(X,y,W)
49             sleep(.1)
50         end
51
52         # Find mean of each cluster
53         for c in 1:k
54             W[c,:] = median(X[y.==c,:],dims=1)
55         end
56
57         # Optionally visualize the algorithm steps
58         if doPlot && d == 2
59             clustering2Dplot(X,y,W)
60             sleep(.1)
61         end
62
63         # Compute the kmeans error
64         l1error = l1KMediansError(X, y, W);
65
66         @printf("Running k-means, changes = %d, k-means error = %.4f \n", changes, l1error)
67     end
68
69     function predict(Xhat)
70         (t,d) = size(Xhat)
71
72         D = l1Distance(Xhat,W)
73         D[findall(isnan.(D))] .= Inf
74
75         yhat = zeros{Int64,t}
76         for i in 1:t
77             (~,yhat[i]) = findmin(D[i,:])
78         end
79         return yhat
80     end
81
82     return PartitionModel(predict,y,W)
83 end
84

```

Figure 7: L1 K-Medians implementation.

2.2 Regularization and Hyper-Parameter Tuning

If you run the script *example_nonLinear* (from Julia's REPL), it will:

1. Load a one-dimensional regression dataset.
2. Fit a least-squares linear regression model.
3. Draw a figure showing the training/testing data and what the model looks like.

Unfortunately, this is not a great model of the data, and the figure shows that a linear model with a y-intercept of 0 is probably not suitable.

1. Update the *leastSquares* function so that it includes a *y-intercept* variable w_0 , and makes predictions using the model

$$y^i = w^T x^i + w_0.$$

Hand in your new function and the updated plot.

Answer: The bias (y-intercept) can be added by adding a column of 1's to the input features. The updated function is shown in Figure 8, and the updated plot is shown in Figure 9.

```
1  include("misc.jl")
2
3  function leastSquares(X,y)
4
5      # Add a columns of 1's to X for the y intercept
6      (n, d) = size(X)
7      X = hcat(ones(n, 1), X)
8
9      # Find regression weights minimizing squared error
10     w = (X'*X)\(X'*y)
11
12     # Make linear prediction function
13     predict(Xtilde) = hcat(ones(size(Xtilde)[1], 1), Xtilde)*w
14
15     # Return model
16     return LinearModel(predict,w)
17 end
```

Figure 8: Linear regression implementation with bias.

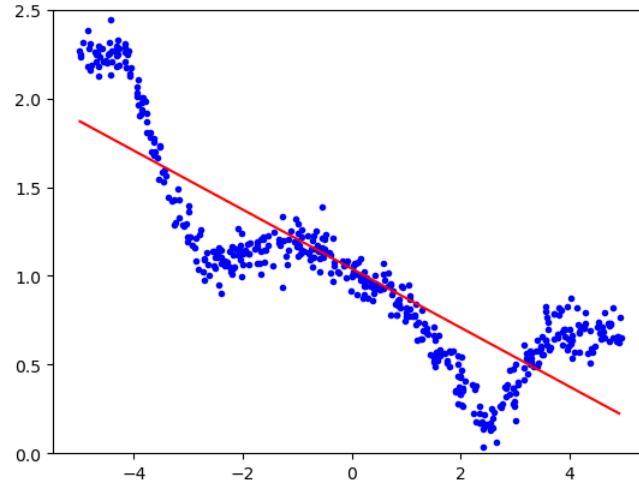


Figure 9: Updated plot with learned y intercept.

2. Allowing a non-zero y-intercept improves the prediction substantially, but the model still seems sub-optimal because there are obvious non-linear effects. Write a function called *leastSquaresRBFL2* that implements *least squares using Gaussian radial basis functions (RBFs) and L2-regularization*.

You should start from the *leastSquares* function and use the same conventions: n refers to the number of training examples, d refers to the number of features, X refers to the data matrix, and y refers to the targets. Use Z as the data matrix after the change of basis, σ as the standard deviation in the Gaussian, and λ as the regularization parameter. Note that you'll have to add two additional input arguments (λ for the regularization parameter and σ for the Gaussian RBF variance) compared to the *leastSquares* function. To make your code easier to understand/debug, you may want to define a new function *rbfBasis* which computes the Gaussian RBFs for a given training set, testing set, and σ value. [Hand in your function and the plot generated with \$\lambda = 1\$ and \$\sigma = 1\$.](#)

Note: the *distancesSquared* function in *misc.jl* is a vectorized way to quickly compute the squared Euclidean distance between all pairs of rows in two matrices.

Answer: The implementation for *rbfBasis* is shown in Figure 10, and the implementation for *leastSquaresRBFL2* is shown in Figure 11. Lastly, the regression plot is shown in Figure 12.

```

3  function rbfBasis(X1, X2, sigma)
4
5      # Create the matrix containing L2 distance squared
6      D = distancesSquared(X1, X2);
7
8      # Compute the basis
9      Z = exp.(-1 .* D / (2*sigma^2));
10
11     Z;
12 end

```

Figure 10: rbfBasis implementation.

```

14 using LinearAlgebra
15
16 function leastSquaresRBFL2(X,y; lambda=1, sigma=1)
17
18     (n, d) = size(X);
19
20     # Change the input samples basis to RBF
21     Z = rbfBasis(X, X, sigma);
22
23     # Find regression weights using RBF basis and L2 norm (Solution found form 340)
24     V = inv(Z'*Z + lambda * I(size(Z)[2])) * Z' * y;
25
26     # Make linear prediction function
27     function predict(Xtilde)
28         Ztilde = rbfBasis(Xtilde, X, sigma);
29         ytilde = Ztilde * V;
30         ytilde;
31     end
32
33     # Return model
34     return RBFModel(predict, V, X, sigma)
35 end

```

Figure 11: leastSquaresRBFL2 implementation.

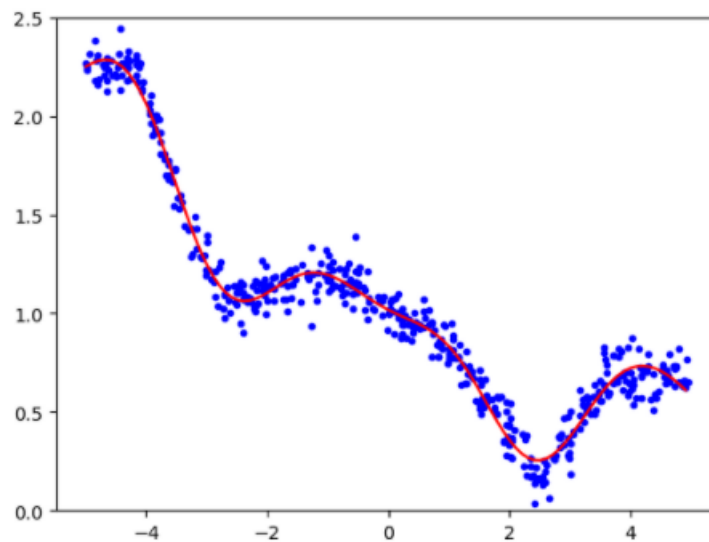


Figure 12: Regression plot with leastSquaresRBFL2.

3. Modify the script to split the training data into a “train” and “validation” set (you can use half the examples for training and half for validation), and use these to select λ and σ . [Hand in your modified script and the plot you obtain with the best values of \$\lambda\$ and \$\sigma\$.](#)

Answer: Code was added to split data into training and validation sets, a grid search was then performed over sigma and lambda values. Mean Squared Error was used to find the best sigma and lambda on the validation set. A lambda value of 1e-5 and sigma of 0.6 had the best performance. The code is shown in Figure 13. Performance on validation set is shown in Figure 14, and the performance on the whole dataset is shown in Figure 15.

```
5 data = load("basisData.jld")
6 (X,y) = (data["X"],data["y"])
7
8 # Compute number of training examples and number of features
9 (n,d) = size(X)
10
11 # Shuffle data
12 shuffle_perm = randperm(n);
13 X = X[shuffle_perm];
14 y = y[shuffle_perm];
15
16 # Split the data into training and validation
17 X_train = X[1:floor(Int, n/2), :];
18 y_train = y[1:floor(Int, n/2), :];
19 X_val = X[floor(Int, n/2)+1:end, :];
20 y_val = y[floor(Int, n/2)+1:end, :];
21
22 # Fit least squares model
23 include("leastSquaresRBFL2.jl")
24
25 sigma_vals = 1e-5:0.1:5;
26 lambda_vals = 1e-5:0.1:5;
27 lowest_mse = 9999999;
28 for sigma_val in sigma_vals
29     for lambda_val in lambda_vals
30         model = leastSquaresRBFL2(X_train, y_train, lambda=lambda_val, sigma=sigma_val);
31         yhat = model.predict(X_val);
32
33         # Compute MSE (mean squared error)
34         mse = sum((yhat - y_val).^2) / (n/2);
35         if mse < lowest_mse
36             global lowest_mse = mse;
37             global chosen_model = model;
38             global chosen_sigma = sigma_val;
39             global chosen_lambda = lambda_val
40         end
41     end
42 end
43
44 @printf("MSE: %.4f with lambda: %.6f and sigma: %.6f", lowest_mse, chosen_lambda, chosen_sigma);
45
46
47 # Plot the validation performance
48 using PyPlot
49 figure()
50 plot(X_val, y_val, "b.")
51 Xhat = minimum(X_val):maximum(X_val)
52 Xhat = reshape(Xhat,length(Xhat),1) # Make into an n by 1 matrix
53 yhat = chosen_model.predict(Xhat)
54 plot(Xhat[:,yhat,"r")
55 ylim((0,2.5))
56 display(gcf())
57
```

Figure 13: Script used to find best lambda and sigma.

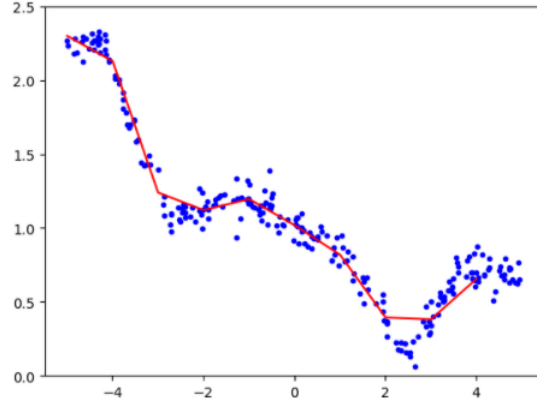


Figure 14: Validation plot with sigma=0.6 and lambda 1e-5.

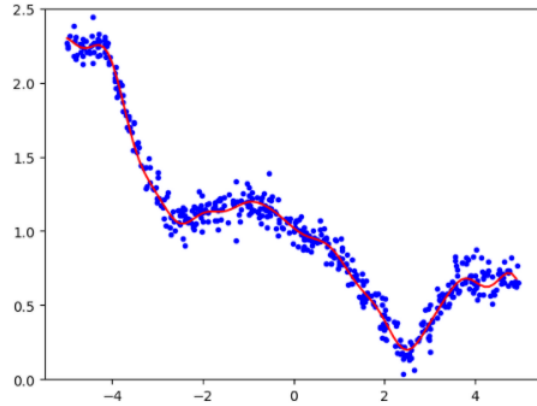


Figure 15: Whole dataset (test) plot with sigma=0.6 and lambda 1e-5.

4. Consider a model combining the first two parts of this question,

$$y^i = w^T x^i + w_0 + v^T z^i,$$

where z^i is the Gaussian RBFs and v is a vector of regression weights for those features. Suppose that we first fit w and w_0 assuming that $v = 0$ (Part 1 of this question), and then we fit v with w and w_0 fixed (you could use your code for Part 2 to do this by modifying the y^i values). Why would someone want to do this?

Hint: think about how this model would behave with $x^i = 10$.

Answer: By having an approach like this, we allow the model to revert to linear regression when we are away from training data. Otherwise, the model would just return 0. To illustrate this, Figure 16 shows that the RBF model predicts 0 for values away from the training data, while the model using the combination of linear regression and RBF reverts back to linear regression as shown in Figure 17. This is because when we are far away from the Gaussian bumps, the second term becomes 0 and only the linear regressoin part remains.

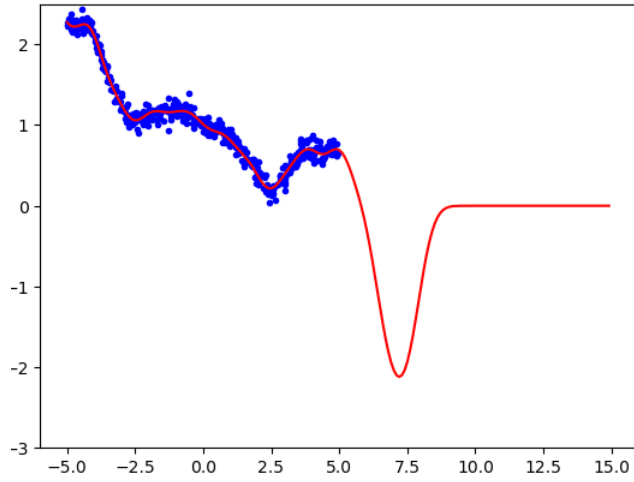


Figure 16: Test sample away from training set with the RBF-only model.

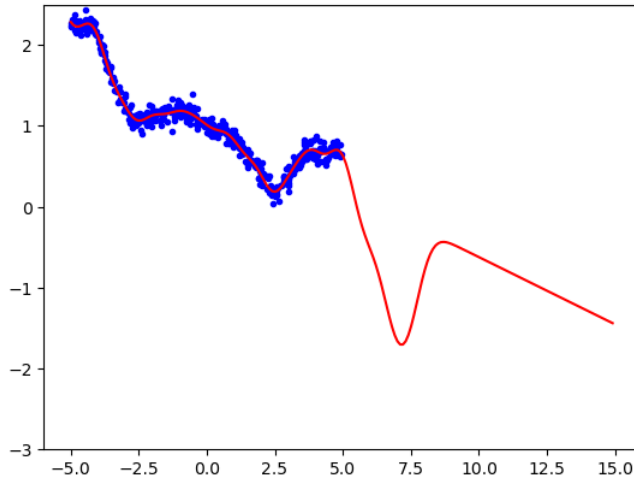


Figure 17: Test sample away from training set with the combined model.

2.3 Multi-Class Logistic Regression

The script *example_multiClass.jl* loads a multi-class classification dataset and fits a “one-vs-all” logistic regression classifier using the *findMin* gradient descent implementation, then reports the validation error and shows a plot of the data/classifier. The performance on the validation set is ok, but could be much better. For example, this classifier never predicts some of the classes.

Using a one-vs-all classifier hurts performance because the classifiers are fit independently, so there is no attempt to calibrate the columns of the matrix W . An alternative to this independent model is to use the softmax probability,

$$p(y^i|W, x^i) = \frac{\exp(w_{y^i}^\top x^i)}{\sum_{c=1}^k \exp(w_c^\top x^i)}.$$

Here c is a possible label and w_c is column c of W . Similarly, y^i is the training label, w_{y^i} is column y^i of W . The loss function corresponding to the negative logarithm of the softmax probability is given by

$$f(W) = \sum_{i=1}^n \left[-w_{y^i}^\top x^i + \log \left(\sum_{c'=1}^k \exp(w_{c'}^\top x^i) \right) \right].$$

Make a new function, *softmaxClassifier*, which fits W using the softmax loss from the previous section instead of fitting k independent classifiers. [Hand in the code and report the validation error.](#)

Hint: you can use the *derivativeCheck* option when calling *findMin* to help you debug the gradient of the softmax loss. Also, note that the *findMin* function treats the parameters as a vector (you may want to use *reshape* when writing the softmax objective).

Answer: The implementation is shown in Figure 19. A validation error of 0.026 was achieved. The classification regions are also shown in Figure 18.

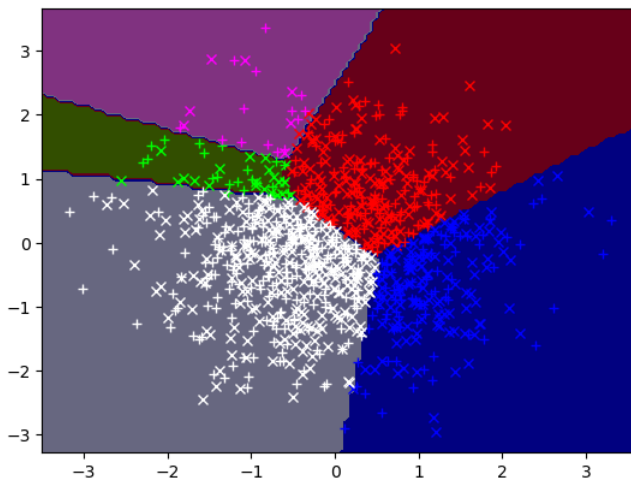


Figure 18: The achieved classification boundaries with Softmax classifier - val error = 0.026.

```

55 function softmaxObj(W, X, y, d, k)
56     # Reshape the weights
57     W = reshape(W, d, k);
58
59     n = size(y)[1];
60
61     # Compute intermediate components
62     XW = X*W;
63     exp_a = exp(X*W);
64     denom_norm = sum(exp_a, dims=2);
65     s = broadcast(/, exp_a, denom_norm);
66
67     # Objective
68     lse = log(denom_norm);
69     XW_yi = [XW[i, y[i]] for i in 1:n];
70     f = sum(-1*XW_yi + lse);
71
72     # Gradient (non-vectorized)
73     g = zeros(d, k)
74     for c in 1:k
75         for j in 1:d
76             for i in 1:n
77                 if y[i] == c
78                     g[j, c] += X[i, j] * (s[i, c] - 1)
79                 else
80                     g[j, c] += X[i, j] * s[i, c]
81                 end
82             end
83         end
84     end
85
86     g = reshape(g, d*k, 1)
87
88     return (f,g)
89 end
90
91 # Multi-class classifier with Softmax
92 function softmaxClassifier(X, y)
93     (n, d) = size(X);
94
95     # Initialize weights
96     k = maximum(y)
97     W = zeros(d*k, 1);
98
99     funObj(W) = softmaxObj(W, X, y, d, k);
100
101     W = findMin(funObj, W, verbose=false, derivativeCheck=true)
102     W = reshape(W, d, k)
103
104     # Make linear prediction function
105     predict(Xhat) = mapslices(argmax, Xhat*W, dims=2)
106
107     return LinearModel(predict, W)
108 end

```

Figure 19: Implementation for multi-class classifier with Softmax.

CITATION: The gradient implementation was first derived then checked to ensure it matched CPSC340 notes at <https://www.students.cs.ubc.ca/~cs-340/homework/a4.pdf>

3 Calculation Questions

3.1 Matrix Notation, Quadratics, Convexity, and Gradients

This question reviews several computational tools covered in CPSC 340. You may also find some of the notes on the course webpage useful.

1. Consider the function

$$f(x) = \sum_{i=1}^n \sum_{j=1}^n a_{ij} x_i x_j + \sum_{i=1}^n b_i x_i + c,$$

where x is a vector of length n with elements x_i , b is a vector of length n with elements b_i , and A is an $n \times n$ matrix with elements a_{ij} (not necessarily symmetric). Write this function in matrix notation (so it uses A and b , and does not have summations or references to indices i).

Answer: Help from course notes at <https://www.cs.ubc.ca/~schmidtm/Courses/Notes/linearQuadraticGradients.pdf> was used for this question.

$$f(x) = \sum_{i=1}^n \sum_{j=1}^n a_{ij} x_i x_j + \sum_{i=1}^n b_i x_i + c$$

$$f(x) = \sum_{i=1}^n \sum_{j=1}^n x_i a_{ij} x_j + \sum_{i=1}^n b_i x_i + c$$

$$f(x) = \sum_{i=1}^n \sum_{j=1}^n x^T A x + \sum_{i=1}^n b_i x_i + c$$

$$f(x) = x^T A x + b^T x + c$$

2. Write the gradient of f from the previous question in matrix notation.

Answer: As suggested in <https://www.cs.ubc.ca/~schmidtm/Courses/Notes/linearQuadraticGradients.pdf>, the non-matrix forms are used first to make this process is easier.

Let's take care of $\sum_{i=1}^n \sum_{j=1}^n a_{ij} x_i x_j$ first:

$$g(x) = \sum_{i=1}^n \sum_{j=1}^n a_{ij} x_i x_j$$

$$g(x) = \sum_{i=1}^n (a_{ii} x_i^2 + \sum_{j \neq i} x_i a_{ij} x_j)$$

$$\frac{\partial}{\partial x_k} \left[\sum_{i=1}^n (a_{ii} x_i^2 + \sum_{j \neq i} x_i a_{ij} x_j) \right] = 2a_{kk} x_k + \sum_{j \neq i} x_j a_{jk} + \sum_{j \neq i} a_{kj} x_j$$

$$\frac{\partial}{\partial x_k} \left[\sum_{i=1}^n (a_{ii} x_i^2 + \sum_{j \neq i} x_i a_{ij} x_j) \right] = \sum_{j=1}^n x_j a_{jk} + \sum_{j=1}^n a_{kj} x_j$$

Now, transforming the above into matrix form, we get:

$$\nabla g(x) = A^T x + A x$$

The second term in $f(x)$ is a lot simpler since it's very similar to simple multiplication:

$$\begin{aligned}h(x) &= b^T x \\ \nabla h(x) &= b\end{aligned}$$

Since the scalar can be ignored, the final answer is:

$$\nabla f(x) = A^T x + Ax + b = (A^T + A)x + b$$

3. Give a linear system whose solution gives a minimum value of f in terms of x , in the case where A is symmetric and positive semi-definite (which implies that f is convex).

Answer: Since the function is convex and differentiable, setting its gradient to 0 and solving for x , then computing the function with this x would give the minimum value of f .

$$\begin{aligned}\nabla f(x) &= A^T x + Ax + b = (A^T + A)x + b = 0 \\ \nabla f(x) &= 2Ax + b = 0 \\ 2Ax &= -b\end{aligned}$$

Now, solving for x and plugging it into the formula for f , we get the minimum value.

4. Consider weighted linear regression with an L2-regularizer with regularization strength $1/\sigma^2$,

$$f(w) = \frac{1}{2} \sum_{i=1}^n v^i (y^i - w^T x^i)^2 + \frac{1}{2\sigma^2} \sum_{j=1}^d w_j^2,$$

where $\sigma > 0$ and we have a vector v of length n containing the elements v^i . The other symbols are our standard regression notation. [Write this function in matrix notation.](#)

Note: you use X as the matrix containing x^i as the rows, y as the vector containing the elements y^i , and V as a diagonal matrix containing the vector v along the diagonal.

Answer: Citation: To solve this, the following PDF file was used <https://www.stat.cmu.edu/~cshalizi/mreg/15/lectures/24/lecture-24--25.pdf>.

$$\begin{aligned} f(w) &= \frac{1}{2} \sum_{i=1}^n v^i (y^i - w^T x^i)^2 + \frac{1}{2\sigma^2} \sum_{j=1}^d w_j^2 \\ f(w) &= \frac{1}{2} \sum_{i=1}^n v^i (y^i - w^T x^i)^2 + \frac{1}{2\sigma^2} * w^T w \\ f(w) &= \frac{1}{2} (y - Xw)^T V (y - Xw) + \frac{1}{2\sigma^2} * w^T w \\ f(w) &= \frac{1}{2} (y^T - w^T X^T) (Vy - VXw) + \frac{1}{2\sigma^2} * w^T w \\ f(w) &= \frac{1}{2} (y^T Vy - y^T VXw - w^T X^T Vy + w^T X^T VXw) + \frac{1}{2\sigma^2} * w^T w \\ f(w) &= \frac{1}{2} (y^T Vy - y^T VXw - w^T X^T Vy + w^T X^T VXw + \frac{1}{\sigma^2} * w^T w) \end{aligned}$$

5. Assuming that we have $v^i \geq 0$ for all i , [show that \$f\$ from the previous part is convex.](#)

Answer: To show convexity of f , we can show that its Hessian is non-negative (all elements are non-negative). We have the gradient (derivation is shown in the next part) as $\nabla f(w) = -X^T Vy + X^T VXw + \frac{1}{\sigma^2} w$. The Hessian is then:

$$\nabla^2 f(w) = (X^T VX)^T + \frac{1}{\sigma^2} I = X^T VX + \frac{1}{\sigma^2} I$$

Now since σ^2 and diagonal entries of V are non-negative, Hessian's entries are all non-negative, which implies that f is convex.

6. Assuming that we have $v^i \geq 0$ for all i , [give a linear system whose solution gives a minimum value of \$f\$ in terms of \$w\$.](#)

Answer: Need to find the gradient first (using properties of V being diagonal and the quadratic gradients we found in the previous question):

$$\begin{aligned} \nabla f(w) &= \frac{1}{2} (-(y^T VX)^T - X^T Vy + (X^T VX + (X^T VX)^T)w + \frac{2}{\sigma^2} w) \\ \nabla f(w) &= \frac{1}{2} (-2X^T Vy + 2X^T VXw + \frac{2}{\sigma^2} w) \\ \nabla f(w) &= -X^T Vy + X^T VXw + \frac{1}{\sigma^2} w \end{aligned}$$

Now setting this to zero would allow us to find the minimum value of function since it is convex:

$$\begin{aligned} -X^T V y + X^T V X w + \frac{1}{\sigma^2} w &= 0 \\ (X^T V X + \frac{1}{\sigma^2} I) w &= X^T V y \end{aligned}$$

Plugging the solution of the above equation for w and plugging it into $f(w)$ would give us the minimum value of f .

7. If we fit a linear classifier with the exponential loss (used in older boosting algorithms), it would take the form

$$f(w) = \sum_{i=1}^n \exp(-y^i w^T x^i).$$

Derive the gradient of this loss function.

Answer: If w is d -dimensional, we have:

$$\nabla f(w) = \begin{bmatrix} \frac{df}{dw_1} \\ \frac{df}{dw_2} \\ \cdot \\ \cdot \\ \cdot \\ \frac{df}{dw_d} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n -\exp(-y^i w^T x^i) y^i x_1^i \\ \sum_{i=1}^n -\exp(-y^i w^T x^i) y^i x_2^i \\ \cdot \\ \cdot \\ \cdot \\ \sum_{i=1}^n -\exp(-y^i w^T x^i) y^i x_d^i \end{bmatrix}$$

Therefore, we have:

$$\frac{df}{dw_d} = \sum_{i=1}^n -e^{(-y^i w^T x^i)} y^i x_d^i$$

$$\nabla f(w) = \sum_{i=1}^n -e^{(-y^i w^T x^i)} y^i x^i$$

8. The support vector regression objective function is

$$f(w) = \sum_{i=1}^n \max\{0, |w^T x^i - y^i| - \epsilon\} + \frac{\lambda}{2} \|w\|^2.$$

where ϵ is a non-negative hyper-parameter. It is similar to the L1 robust regression loss, but where there is no penalty for being within ϵ of the prediction (which can reduce overfitting). [Show that this non-differentiable function is convex.](#)

Answer: Using CPSC340 notes on convexity, we have the following rules:

- (a) Norms and squared norms are convex.
- (b) The sum of convex functions is a convex function.
- (c) A convex function multiplied by non-negative constant is convex.
- (d) The max of convex functions is convex.

$\ w\ ^2$	is convex because of (a)
$\lambda/2 * \ w\ ^2$	is convex because of (c)
$ w^T x^i - y^i $	is convex because of (a)
$ w^T x^i - y^i - \epsilon$	is convex because of (b)
$\max\{0, w^T x^i - y^i - \epsilon\}$	is convex because of (d)
$f(w) = \sum_{i=1}^n \max\{0, w^T x^i - y^i - \epsilon\} + \frac{\lambda}{2} \ w\ ^2$	is convex because of (b)

3.2 MAP Estimation

In 340, we showed that under the assumptions of a Gaussian likelihood and Gaussian prior,

$$y^i \sim \mathcal{N}(w^\top x^i, 1), \quad w_j \sim \mathcal{N}\left(0, \frac{1}{\lambda}\right),$$

that the MAP estimate is equivalent to solving the L2-regularized least squares problem

$$f(w) = \frac{1}{2} \sum_{i=1}^n (w^\top x^i - y^i)^2 + \frac{\lambda}{2} \sum_{j=1}^d w_j^2,$$

in the “loss plus regularizer” framework. For each of the alternate assumptions below, write it in the “loss plus regularizer” framework (simplifying as much as possible):

1. Gaussian likelihood with a separate variance σ_i^2 for each training example, and Laplace prior with a separate variance $1/\lambda_j$ for each variable,

$$y^i \sim \mathcal{N}(w^\top x^i, \sigma_i^2), \quad w_j \sim \mathcal{L}\left(0, \frac{1}{\lambda_j}\right).$$

Answer: For all answers I’m assuming IID training samples. Let’s look at the prior first:

$$\begin{aligned} p(w) &= \prod_{j=1}^d p(w_j) \propto \prod_{j=1}^d \exp\left(-\frac{|w_j|}{\frac{1}{\lambda_j}}\right) = \prod_{j=1}^d \exp(-\lambda_j |w_j|) \\ -\log(p(w)) &= \sum_{j=1}^d \lambda_j |w_j| + \text{constant} \end{aligned}$$

This looks like a L1 norm regularizer with different regularization ratio for each sample. Now let’s look at the likelihood term:

$$\begin{aligned} p(y|W, X, \sigma) &\propto \prod_{i=1}^n \exp\left(-\frac{1}{2} \left(\frac{w^\top x^i - y^i}{\sigma_i}\right)^2\right) = \prod_{i=1}^n \exp\left(-\frac{(w^\top x^i - y^i)^2}{2\sigma_i^2}\right) \\ -\log(p(y|W, X, \sigma)) &= \frac{1}{2} \sum_{i=1}^n \frac{1}{\sigma_i^2} (w^\top x^i - y^i)^2 \end{aligned}$$

Therefore, the final answer is:

$$f(w) = \frac{1}{2} \sum_{i=1}^n \frac{1}{\sigma_i^2} (w^\top x^i - y^i)^2 + \sum_{j=1}^d \lambda_j |w_j|$$

2. Robust student- t likelihood and Gaussian prior centered at u .

$$p(y^i|x^i, w) = \frac{1}{\sqrt{\nu} B\left(\frac{1}{2}, \frac{\nu}{2}\right)} \left(1 + \frac{(w^\top x^i - y^i)^2}{\nu}\right)^{-\frac{\nu+1}{2}}, \quad w_j \sim \mathcal{N}\left(u_j, \frac{1}{\lambda}\right),$$

where u is $d \times 1$, B is the “Beta” function, and the parameter ν is called the “degrees of freedom”.¹

¹This likelihood is more robust than the Laplace likelihood, but leads to a non-convex objective.

Answer: Let's look at the prior first:

$$p(w) = \prod_{j=1}^d p(w_j) \propto \prod_{j=1}^d \exp\left(-\frac{\lambda}{2}(w_j - u_j)^2\right)$$

$$-\log(p(w)) = \frac{\lambda}{2} \sum_{j=1}^d (w_j - u_j)^2 + \text{constant}$$

This looks like a shifted L2 regularizer. Now let's look at the likelihood term:

$$p(y|W, X) \propto \prod_{i=1}^n p(y^i|x^i, w) = \left(1 + \frac{(w^T x^i - y^i)^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

$$-\log(p(y|W, X, \sigma)) = \frac{\nu+1}{2} \sum_{i=1}^n \log\left(1 + \frac{(w^T x^i - y^i)^2}{\nu}\right) + \text{constant}$$

Therefore, the final answer is:

$$f(w) = \frac{\nu+1}{2} \sum_{i=1}^n \log\left(1 + \frac{(w^T x^i - y^i)^2}{\nu}\right) + \frac{\lambda}{2} \sum_{j=1}^d (w_j - u_j)^2$$

3. We use a Poisson-distributed likelihood (for the case where y_i represents counts), and we use a uniform prior for some constant κ ,

$$p(y^i|x^i, w) = \frac{\exp(y^i w^T x^i) \exp(-\exp(w^T x^i))}{y^i!}, \quad p(w_j) \propto \kappa.$$

(This prior is “improper” since $w \in \mathbb{R}^d$ but it doesn't integrate to 1 over this domain.)

Answer: Let's look at the prior first:

$$-\log(p(w)) = -\sum_{j=1}^d \log(\kappa) = \text{constant}$$

Therefore, we don't have a regularizer. Now let's look at the likelihood term:

$$p(y|W, X) \propto \prod_{i=1}^n p(y^i|x^i, w) = \prod_{i=1}^n \frac{\exp(y^i w^T x^i) \exp(-\exp(w^T x^i))}{y^i!}$$

$$-\log(p(y|W, X, \sigma)) = \sum_{i=1}^n -y^i w^T x^i + \exp(w^T x^i) + \sum_{i=1}^n \log(y^i!) = \sum_{i=1}^n -y^i w^T x^i + \exp(w^T x^i) + \text{constant}$$

Therefore, the final answer is:

$$f(w) = \sum_{i=1}^n (-y^i w^T x^i + \exp(w^T x^i))$$

3.3 Machine Learning Model Memory and Time Complexities

Answer the following questions using big-O notation, and a brief explanation. Your answers may involve n , d , and perhaps additional quantities defined in the question. As an example, the (linear) least squares model has $O(d)$ parameters, requires $O(d)$ time for prediction, and requires $O(nd^2 + d^3)$ time to train.²

1. What is the storage space required for the k -means clustering algorithm?

Answer: $O(kd)$ because we need to keep k cluster means for each dimension in d . We also keep labels for each cluster which is $O(k)$.

2. What is the cost of clustering t examples using an already-trained k -means model?

Answer: $O(tkd)$ because we need to compare each test sample with each mean for each dimension.

3. What is the training time for linear regression with L2-regularization?

Answer: $O(nd^2 + d^3)$ because the least squares is obtained with $w = (X^T X + \lambda I)^{-1} (X^T y)$ with $X^T y$ being $O(d)$, constructing $X^T X$ being $O(nd^2)$ and solving the equation being $O(d^3)$. Note that $O(d)$ can be ignored since other terms dominate it. Citation: CPSC340 Notes.

4. What is the prediction cost for linear regression with L2-regularization on t test examples?

Answer: $O(td)$ since we have to do the following matrix multiplication WX .

²In this course, we assume matrix operations have the “textbook” cost where the operations are implemented in a straightforward way with “for” loops. For example, we’ll assume that multiplying two $n \times n$ matrices or computing a matrix inverse simply costs $O(n^3)$, rather than the $O(n^\omega)$ where ω is closer to 2 as discussed in CS algorithm courses.

5. What is the storage space required for linear regression with Gaussian RBFs as features?

Answer: $O(nd)$ since we need all training samples to construct Z .

6. What is the prediction time for linear regression with Gaussian RBFs as features on t test examples? You can use σ^2 as the variance of the Gaussian RBFs.

Answer: $O(tnd)$ since we need to construct Z which requires finding L2 distance between each t and each n which has dimension d . This dominates the matrix multiplication required for prediction.

7. What is the cost of evaluating the support vector regression objective function?

Answer: $O(nd)$ since we have $O(nd)$ for the evaluation of XW term (in $\sum_{i=1}^n \max(0, 1 - y_i w^T x_i + \frac{\lambda}{2} \|W\|^2)$) and $O(d)$ for the L2 norm which is dominated by the other term.

8. What is the cost of trying to minimize the exponential loss regression loss by running t iterations of gradient descent?

Answer: $O(ndt)$ with t being the number of iterations because the XW term in $f(w) = \sum_{i=1}^n \exp(-y^i w^T x^i)$ needs $O(nd)$.

9. What is the cost of trying to minimize the exponential loss by running t iterations of stochastic gradient descent?

Answer: $O(td)$ where t is the number of iterations. This is because in SGD we only use one sample per iteration which has dimension d and $w^T x^i$ only needs $O(d)$.

Hint: many people got very-low grades on this question last year. If you are not sure how to answer questions like this, get help!

CPSC 440/540 Machine Learning (January-April, 2022)

Assignment 2 (due Friday February 11th at midnight)

The assignment instructions are the same as for the previous assignment, but for this assignment you can work in groups of 1-2. However, please only hand in one assignment for the group.

1. Name(s):

Answer: Masoud Mokhtari

2. Student ID(s):

Answer: 14186167

1 Bernoulli Distributions

1.1 Inference

Consider a Bernoulli distribution with $\theta = 0.12$.

1. Suppose we generate 10 IID samples $\{x^1, x^2, \dots, x^{10}\}$ according to this model. What is the probability that all 10 samples are equal to 0?

Answer: With the IID assumption, we have:

$$p(x^1, x^2, \dots, x^{10}) = \prod_{i=1}^{10} p(x^i | \theta) = \theta^{\sum_{i=1}^{10} I[x^i=1]} (1 - \theta)^{\sum_{i=1}^{10} I[x^i=0]}$$

Therefore, for all samples to be 0:

$$p(x^1 = 0, x^2 = 0, \dots, x^{10} = 0) = (1 - \theta)^{10} = 0.88^{10} = 0.2785$$

2. What is the probability that exactly one sample is equal to 1, and the other 9 are equal to 0?

Answer: Again, with the IID assumption, we first have to choose 1 of 10 samples to be 1:

$$p\left(\sum_{i=1}^{10} x^i = 1 | \theta\right) = \binom{10}{1} * \theta^1 * (1 - \theta)^9 = 10 * 0.12 * 0.88^9 = 0.3798$$

3. Suppose we need to base decisions on samples from this dataset, and so we want collect a set of 1000 samples. Would it make sense to chose the decoding of the 1000 samples (the “most likely” samples to be seen) to make decisions? Explain why or why not?

Answer: No it does not make sense to use decoding for decision making because decoding does not represent typical behaviour of the data. For example, in the case of this question where $\theta = 0.12$, decoding for the 1000 samples would cause them all to be chosen as 0 even though there is a 12 percent chance that some samples are 1. Therefore, this behaviour is not representative of the data. In other words, while decoding has the highest probability, this probability can be very low.

4. Write a function that generates t IID examples from this distribution, that uses Julia's `rand()` function as the source of IID randomness (assuming that each call to `rand()` returns a number uniformly-distributed between 0 and 1). [Hand in the code](#).

Answer:

```
function sampleBernoulli(t, theta=0.12)

    # Initialize array holding samples
    samples = zeros(t);

    # Produce t samples
    for i in 1:t
        uniform_sample = rand();

        if uniform_sample <= theta
            samples[i] = 1;
        end
    end

    return samples
end
```

Figure 1: Julia code for sampling t samples from Bernoulli using `rand()` function

5. Consider a game where you get \$1 if a sample from this distribution is 0, and lose \$5 if the sample is 1. What is the expected \$ value you get from playing this game?

Answer: Using the expected value formula:

$$E[f(x)] = P(x = 1|\theta)f(x = 1) + P(x = 0|\theta)f(x = 0) = 0.12 * (-5) + 0.88 * 1 = 0.28$$

Therefore, the expected \$ value is 28 cents.

6. A way to approximate the answer of the previous question is given by

$$\frac{1}{t} \sum_{i=1}^t f(x^i),$$

where each x^i is one of t IID sample and f gives the \$ value for that sample (either 1 or -5). Implement this approximation, and hand in a plot the of value of this approximation as you vary the number of samples t from 1 to 100,000.

Answer: As seen in Figure 2, as we increase the number of samples, our approximation converges to the computed value in the previous question.

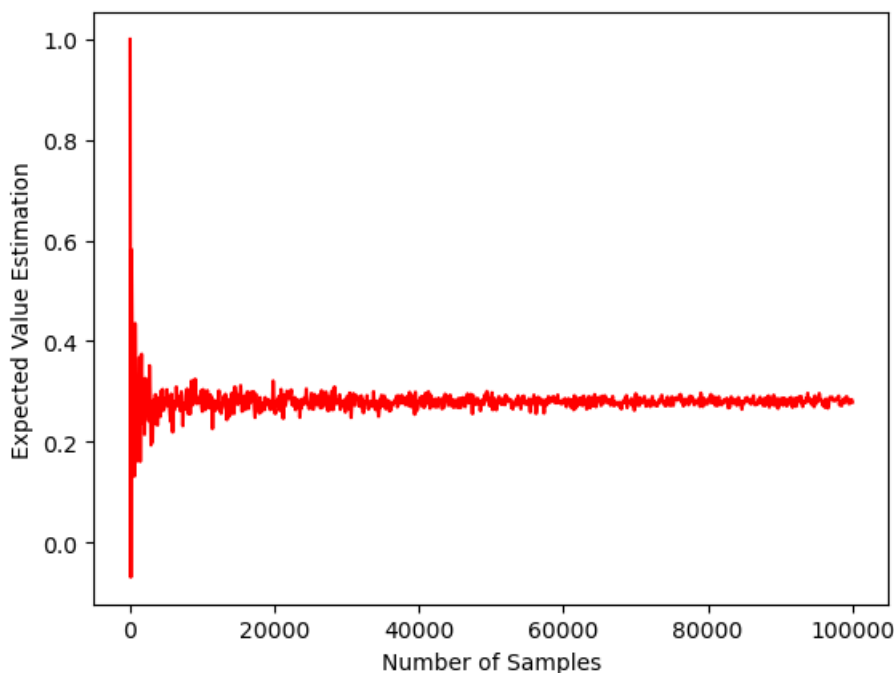


Figure 2: Approximation of expected value

7. Suppose that you wanted to compute the expected number of samples from this Bernoulli that you would need to generate before seeing 10 values of 1. It possible to compute this expectation exactly, but suppose we really like sampling and we are too lazy to do the exact calculation. Give an approximation of this quantity based on generating samples, similar to the one used in Part 6.

Answer: We can do this by defining a new function $g(\theta)$ that generates samples from Bernoulli with θ until 10 successes are observed and returns the number of samples required to achieve this. Now if we

call this function n times, we can approximate the expected value as:

$$\frac{1}{n} \sum_{i=1}^n g(\theta)$$

A Julia implementation of this is shown in Figure 3. Using $n = 10000$, the expected value is approximated to be 83.389 samples.

```

1  using JLD
2  using Printf
3
4  include("sampleBernoulli.jl")
5
6  theta = 0.12;
7  total_num_samples = 0
8  num_trials = 10000
9
10 for n in 1:num_trials
11     num_ones = 0
12     num_samples = 0
13     while num_ones < 10
14         sample = sampleBernoulli(1, theta);
15         num_samples += 1
16         if sample[1] == 1
17             num_ones += 1
18         end
19     end
20     global total_num_samples += num_samples
21 end
22
23 @printf("Number of samples to get 10 successes %.4f\n", total_num_samples/num_trials)

```

Figure 3: Code implementation to find the expected number of samples needed to get 10 successes.

8. Suppose that the `rand()` function was broken in the new version of Julia, in the sense that the numbers it returned were found to not look like uniform samples.¹ Also suppose that the `randn()` function, which generates samples from a standard normal distribution, worked well. Describe a way to generate samples from a Bernoulli distribution that relies on the `randn()` function instead of the `rand()` function. Hint: you may to look up the CDF (and its inverse the “quantile” function) of the standard normal distribution.

Answer: While directly using the PMF of the Gaussian is not suitable for this task, we can use its area under the curve (or CDF) to generate Bernoulli samples as it follows a uniform distribution. More specifically, we know that if P is the CDF of X , its inverse $P^{-1}(q)$ is the value x_q such that $P(X \leq x) = q$ where q is the q 'th quantile of P (reference: <https://mitpress.mit.edu/books/series/adaptive-computation-and-machine-learning-series>). Now if we set q to be 0.12 (the θ for the Bernoulli), we get a value for $x_{0.12}$. Now if we sample the Gaussian, for sample values x that satisfy $x \leq x_{0.12}$ we generate success, and for cases that satisfy $x > x_{0.12}$ we generate fail. An implementation for this is shown in Figure 4.

¹Some published research works have had to be revised due to bad random number generators.


```

1  using Distributions
2
3  function sampleBernoulli(t, theta=0.12)
4
5      x_theta = quantile(Normal(), theta)
6
7      # Initialize array holding samples
8      samples = zeros(t);
9
10     # Produce t samples
11     for i in 1:t
12         gaussian_sample = randn();
13
14         if gaussian_sample <= x_theta
15             samples[i] = 1;
16         end
17     end
18
19     return samples
20
21 end

```

Figure 4: Code implementation of generating Bernoulli samples using randn.

1.2 Learning

1. Consider a dataset consisting of binary samples $\{x^1, x^2, \dots, x^n\}$. Consider the binomial likelihood for such a dataset,

$$p(x^1, x^2, \dots, x^n | \theta) = \binom{n}{k} \theta^k (1 - \theta)^{n-k},$$

where $k = \sum_{i=1}^n x^i$ is the number of 1 values in the dataset and $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ is the number of combinations of k items that exist among n items. This likelihood measures the probability of seeing exactly k values of 1 in the dataset, as opposed to the Bernoulli likelihood which measures the likelihood of seeing the precise sequence x^1, x^2, \dots, x^n . **Derive the MAP estimate of θ for this model if we use a beta prior on θ with hyper-parameters α and β .** You can assume that k and $(n - k)$ are both positive. You can also assume that the data is independent of the hyper-parameters if we condition on the parameters, which mathematically is equivalent to $p(X | \theta, \alpha, \beta) = p(X | \theta)$.

Answer: For MAP, we would like to find $\hat{\theta} \in \arg \max_{\theta} P(\theta | X, \alpha)$. For this question, with a Beta prior for the posterior we have:

$$P(\theta | X, \alpha, \beta) \propto P(X | \theta, \alpha, \beta) P(\theta | \alpha, \beta) \propto \binom{n}{k} \theta^k (1 - \theta)^{n-k} \theta^{\alpha-1} (1 - \theta)^{\beta-1}$$

Now, since $\binom{n}{k}$ is a constant that doesn't affect θ :

$$\binom{n}{k} \theta^k (1 - \theta)^{n-k} \theta^{\alpha-1} (1 - \theta)^{\beta-1} \propto \theta^k (1 - \theta)^{n-k} \theta^{\alpha-1} (1 - \theta)^{\beta-1} = \theta^{k+\alpha-1} (1 - \theta)^{n-k+\beta-1}$$

Now let's set $\hat{\alpha} = k + \alpha$ and $\hat{\beta} = n - k + \beta$. This makes the posterior a beta distribution with $\hat{\alpha}$ and $\hat{\beta}$. Now we can take the log of the formula to get:

$$P(\theta | X, \alpha, \beta) \propto (\hat{\alpha} - 1) \log(\theta) + (\hat{\beta} - 1) \log(1 - \theta)$$

Taking the derivative of the above and setting it to 0, then solving for θ would give us the solution:

$$\begin{aligned} (\hat{\alpha} - 1) \frac{1}{\theta} - (\hat{\beta} - 1) \frac{1}{1 - \theta} &= 0 \\ (\hat{\alpha} - 1) - (\hat{\beta} - 1) \frac{\hat{\theta}}{1 - \hat{\theta}} &= 0 \\ (\hat{\beta} - 1) \frac{\hat{\theta}}{1 - \hat{\theta}} &= \hat{\alpha} - 1 \\ \hat{\theta} &= \frac{\hat{\alpha} - 1}{\hat{\alpha} - 1 + \hat{\beta} - 1} = \frac{k + \alpha - 1}{n + \alpha + \beta - 2} \end{aligned}$$

NOTE: the above $\hat{\theta}$ is the correct solution since the second derivative is of the form $-(\hat{\alpha} - 1) \frac{1}{\theta^2} - (\hat{\beta} - 1) \frac{1}{(1 - \theta)^2}$. This is negative since k and $n - k$ are positive. Therefore, the function is concave down and the found θ is the point where we achieve the maximum.

2. We often use Bernoulli distributions as parts of more-complicated distributions. In these cases we often to need to maximize a weighted log-likelihood of the form

$$f(\theta) = \sum_{i=1}^n v^i \log p(x^i | \theta),$$

where each v^i is a non-negative weight for example i . Derive the MLE for this model when we use a Bernoulli likelihood for $p(x^i | \theta)$.

Answer: For MLE, we would like to find $\hat{\theta} \in \arg \max_{\theta} \sum_{i=1}^n v^i \log p(x^i | \theta)$:

$$\begin{aligned} f(\theta) &= \sum_{i=1}^n v^i \log p(x^i | \theta) \\ &= \sum_{i=1}^n v^i [x^i \log(\theta) + (1 - x^i) \log(1 - \theta)] \\ &= (\sum_{i=1}^n v^i I[x^i = 1]) \log(\theta) + (\sum_{i=1}^n v^i I[x^i = 0]) \log(1 - \theta) \end{aligned}$$

Setting the derivative of above to 0 and solving for θ , we get:

$$\begin{aligned} \frac{\sum_{i=1}^n v^i I[x^i = 1]}{\hat{\theta}} - \frac{\sum_{i=1}^n v^i I[x^i = 0]}{1 - \hat{\theta}} &= 0 \\ (\sum_{i=1}^n v^i I[x^i = 1]) - (\sum_{i=1}^n v^i I[x^i = 1])\hat{\theta} &= (\sum_{i=1}^n v^i I[x^i = 0])\hat{\theta} \\ \sum_{i=1}^n v^i I[x^i = 1] &= (\sum_{x^i \neq 1} v^i + \sum_{x^i = 1} v^i)\hat{\theta} \\ \sum_{i=1}^n v^i I[x^i = 1] &= (\sum_{x^i \neq 1} v^i + \sum_{x^i = 1} v^i)\hat{\theta} \\ \hat{\theta} &= \frac{\sum_{i=1}^n v^i I[x^i = 1]}{\sum_{i=1}^n v^i} \end{aligned}$$

Please note that since x^i can only be 0 or 1, we can simplify the above to $\hat{\theta} = \frac{\sum_{i=1}^n v^i x^i}{\sum_{i=1}^n v^i}$.

Now, let's do a second derivative check to see if this is actually the maximum. The second derivative is:

$$-\frac{\sum_{i=1}^n v^i I[x^i = 1]}{\hat{\theta}^2} - \frac{\sum_{i=1}^n v^i I[x^i = 0]}{(1 - \hat{\theta})^2}$$

Therefore the second derivative is negative due to θ and the numerator sums being non-negative.

2 Multivariate and Conditional Bernoullis

2.1 Naive Bayes

If you run the script `example_mnist35_naiveNaiveBayes`, it will load training and test MNIST digits 3 and 5. It will then discretize the features into binary values, and fit a “naive” naive Bayes model. In particular, the naive naive Bayes model is a generative classifier that assumes $p(x_1, x_2, \dots, x_d, y)$ is product of Bernoullis. This model has a high test error, since the features do not affect the predictions.

1. The regular naive Bayes model discussed in class writes the joint probability of a data $\{X, y\}$ as

$$\begin{aligned} p(X, y) &= \prod_{i=1}^n \left[p(y^i) \prod_{j=1}^d p(x_j^i | y^i) \right] \\ &= \prod_{i=1}^n \left[\theta^{y^i} (1 - \theta)^{1-y^i} \prod_{j=1}^d \theta_{jy^i}^{x_j^i} (1 - \theta_{jy^i})^{1-x_j^i} \right], \end{aligned}$$

where we have used a Bernoulli to parameterize $p(y^i)$ and a Bernoulli-like distribution to parameterize the conditionals $p(x_j | y)$. [Show how to derive the MLE for a particular value of \$\theta_{jc}\$.](#)

Answer: Before attempting to find the MLE, let's take the log of the above equation:

$$\begin{aligned} &= \sum_{i=1}^n \left[\log(p(y^i)) + \log \left[\prod_{j=1}^d p(x_j^i | y^i) \right] \right] \\ &= \sum_{i=1}^n \left[\log(p(y^i)) + \sum_{j=1}^d \log(p(x_j^i | y^i)) \right] \\ &= \sum_{i=1}^n \left[y^i \log(\theta) + (1 - y^i) \log(1 - \theta) + \sum_{j=1}^d [x_j^i \log(\theta_{jy^i}) + (1 - x_j^i) \log(1 - \theta_{jy^i})] \right] \end{aligned}$$

Now, to find the MLE for θ_{jc} we have to take the derivative of the above with respect to θ_{jc} then solve for this value when the derivative is zero:

$$\begin{aligned} \frac{d}{d\theta_{jc}} \left(\sum_{i=1}^n \left[y^i \log(\theta) + (1 - y^i) \log(1 - \theta) + \sum_{j=1}^d [x_j^i \log(\theta_{jy^i}) + (1 - x_j^i) \log(1 - \theta_{jy^i})] \right] \right) &= 0 \\ \sum_{i=1}^n I[y^i = c] \left[x_j^i \frac{1}{\theta_{jc}} - \frac{1 - x_j^i}{1 - \theta_{jc}} \right] &= 0 \end{aligned}$$

Please note that the above is true because all terms in the inner sum where $j' \neq j$ are zero:

$$\begin{aligned} n_{x_j=1, y=c} * \frac{1}{\theta_{jc}} - n_{x_j=0, y=c} * \frac{1}{1 - \theta_{jc}} &= 0 \\ n_{x_j=1, y=c} - n_{x_j=1, y=c} * \theta_{jc} &= n_{x_j=0, y=c} * \theta_{jc} \\ \theta_{jc} &= \frac{n_{x_j=1, y=c}}{n_{y=c}} \end{aligned}$$

Where $n_{x_j=0, y=c}$ and $n_{x_j=1, y=c}$ are number of samples where $x_j = 0$ and $x_j = 1$ respectively while $y^i = c$. And $n_y = c$ is all samples where $y^i = c$ regardless of x_j . Lastly, second derivative check:

$$\sum_{i=1}^n I[y^i = c] \left[x_j^i \frac{-1}{\theta_{jc}^2} - \frac{1 - x_j^i}{(1 - \theta_{jc})^2} \right] < 0.$$

- Even though the naive Bayes likelihood has many parameters, the MLE for a particular parameter θ_{jc} does not depend on θ or any $\theta_{j'c'}$ with $j \neq j'$ and $c' \neq c$. Explain why these terms do not appear in the MLE. (You should not use the word “independence” in your answer. Instead explain how the form of the log-likelihood makes them not depend on each other.)

Answer: When we take the log of the likelihood, all individual terms including $p(y)$ and $p(x_j^i|y^i)$ for all i and j become terms that are added to each other. Additionally, we know that taking the derivative of a sum is equal to sum of derivatives. Lastly, the derivative for terms that do not include θ_{jc} is zero, which makes the MLE for θ_{jc} not depend on all those other terms.

- Modify the naive Bayes code to implement the standard naive Bayes classifier. Hand in your code and report the test error that you obtain. Hint: start from the function *naiveNaiveBayes* and replace the $d \times 1$ variable p_x with a $d \times 2$ variable p_{xy} that will represent θ_{jc} . Hint: you can look at subsequent questions to get an idea of what the final error should be.

Answer: The achieved test error is 0.1, and the code is provided in Figure 5.

```

3 function NaiveBayes(X,y)
4     # Implementation of generative classifier,
5     # where Naive Bayes of Bernoullis is used for p(x,y)
6
7     (n,d) = size(X)
8
9     # Compute p(y = 1)
10    p_y = sum(y==1)/n
11
12    # We will store theta_jc in p_xy(j)
13    p_xy = zeros(d, 2)
14    for j in 1:d
15        for c in 1:2
16            p_xy[j, c] = sum(X[y==c,j]==1)/sum(y==c)
17        end
18    end
19
20    function predict(Xhat)
21        (t,d) = size(Xhat)
22        yhat = zeros(t)
23
24        for i in 1:t
25            # p_yx = p_y*prod(p_x) for the appropriate x and y values
26            p_yx = [p_y;1-p_y]
27            for j in 1:d
28                if Xhat[i,j] == 1
29                    p_yx[1] *= p_xy[j, 1]
30                    p_yx[2] *= p_xy[j, 2]
31                else
32                    p_yx[1] *= 1-p_xy[j, 1]
33                    p_yx[2] *= 1-p_xy[j, 2]
34                end
35            end
36            if p_yx[1] > p_yx[2]
37                yhat[i] = 1
38            else
39                yhat[i] = 2
40            end
41        end
42        return yhat
43    end
44
45    return GenericModel(predict)
46 end
47

```

Figure 5: Naive Bayes implementation.

4. Instead of assuming that all variables are independent given the class label, a less-naive model called “chain-augmented naive Bayes” (CANB) assumes that variables follow a Markov chain given the class label. This leads to a joint likelihood of the form

$$p(x_1^i, x_2^i, \dots, x_d^i, y^i) = p(y^i)p(x_1^i | y^i) \prod_{j=2}^d p(x_j^i | x_{j-1}^i, y^i),$$

where features 2: d depend on not only on the class label but also on the value of the feature that comes “before” it in the ordering 1: d . This captures the fact that adjacent pixels in the image tend to be correlated with each other. Using MLE to estimate the parameters of this model leads to test error of 0.24, while if you add Laplace smoothing it achieves a better test error of 0.14 (which is still worse than naive Bayes). On the other hand, adding Laplace smoothing to naive Bayes does not change the test error for this data. [Explain why Laplace smoothing improves the test error of CANB, while for naive Bayes it does not affect the test error.](#)

Answer: Naive Bayes takes the following form: $p(x_1^i, x_2^i, \dots, x_d^i, y^i) = p(y^i)p(x_1^i | y^i) \prod_{j=2}^d p(x_j^i | y^i)$. Therefore, we can see that the only difference between CANB and Naive Bayes in the $p(x_j^i | x_{j-1}^i, y^i)$ terms.

Now, for the Naive Bayes case, as long as for each j , there is one sample where $x_j = 1$ and one sample where $x_j = 0$, laplace smoothing will not have much effect since zero-probability terms are very unlikely (which is the case in our problem). However, for CANB, even if the number of samples where $x_j = 1$ or $x_j = 0$ is not zero, we still require number of samples where $x_{j-1} = 1$ and $x_{j-1} = 0$ to not be zero since otherwise $p(x_j^i | x_{j-1}^i, y^i)$ would be zero, which is an extra constraint. Therefore, for this example, laplace smoothing actually helps CANB more by preventing zero probability terms.

5. With Laplace smoothing, the CANB from the previous question obtains a much-higher test-set likelihood than naive Bayes. In other words, $p(\tilde{X}, \tilde{y})$ for new data is much higher for the CANB model than naive Bayes. [Explain why the CANB model might give worse test error even though it gives a much-better test set likelihood.](#)

Answer: The problem of density estimation is different from classification. For example, Naive Bayes does not have to be a perfect density estimator (i.e. high test likelihood) to make good classifications, it just needs to build enough inductive bias.

In this case, it seems like the dependence between x_j and x_{j-1} decreases the model’s discriminative power since the dependence is not only on y^i anymore. Therefore, while this model is a better density estimator (due to the correlation assumption), the correlation assumption decreases its discriminative power.

2.2 Vector-Quantized Naive Bayes

If you run the script `example_mnist35_logistic`, it will fit a logistic regression model that achieves a lower test error on the MNIST 3 and 5 digits than naive Bayes does. In this question we will consider one way to make naive Bayes less naive in order to improve its performance.

1. It seems reasonable that there would be clusters in the classes. For example, there might several different ways to draw the digit ‘3’. Instead of assuming that the features are independent given the class label, we might instead assume they are independent given the cluster that they are in. Consider a *vector-quantized naive Bayes* (VQNB) that implements this idea:
 - It clusters the *examples associated with each digit* into k clusters, using k-means clustering (so there will be k clusters for each class and $2k$ clusters total). We will use z^i as the cluster number of example i . The value z^i will be from 1 to k , with y^i determining whether we are considering the k “3” clusters or the k “5” clusters.
 - Since we do not know z^i (it is a “latent” variable), we need to use the marginalization rule to consider its possible values. Using the marginalization and then the product rule, we can write the joint probability as

$$\begin{aligned}
 p(x_1^i, x_2^i, \dots, x_d^i, y^i) &= \sum_{z=1}^k p(x_1^i, x_2^i, \dots, x_d^i, y^i, z) \\
 &= \sum_{z=1}^k p(x_1^i, x_2^i, \dots, x_d^i \mid y^i, z) p(y^i, z) \\
 &= \sum_{z=1}^k p(x_1^i, x_2^i, \dots, x_d^i \mid y^i, z) p(z \mid y^i) p(y^i) \\
 &= p(y^i) \sum_{z=1}^k p(z \mid y^i) \prod_{j=1}^d p(x_j^i \mid y^i, z),
 \end{aligned}$$

where the last line is using the independence of the features given the cluster.

Make a version of your naive Bayes code that implements the VQNB method. [Hand in your code and the test error you obtain with this model for \$k = 2\$ through \$k = 5\$.](#) Hints:

- You can use the variable `p_y` as before. You will need a new variable `p_zy` that is $k \times 2$ representing $p(z = c \mid y^i = b)$. The MLE for an element `p_zy[c, b]` of the matrix is (number of times that $y^i = b$ and $z^i = c$) / (number of times $y^i = b$).²
- You can replace `p_xy` with `p_xyz` which will be a d by 2 by k array giving the value $p(x_j^i = 1 \mid y^i = b, z^i = c)$. The MLE for the value `p_xyz[j, b, c]` is (number of times $y^i = b$ and $z^i = c$ and $x_j^i = 1$) / (number of times $y^i = b$ and $z^i = c$).
- To help with debugging, note that you should get the naive Bayes model in the special case of $k = 1$. Further, with this type of model you usually see the biggest performance gain when going from $k = 1$ to $k = 2$.
- You may not be able to exactly match the performance of logistic regression.

Answer: The results are as follows:

- $k=1$ - Error: 0.10
- $k=2$ - Error: 0.07

²We will derive this MLE for k -valued discrete variables in the next part of the course.

- k=3 - Error: 0.06
- k=4 - Error: 0.05
- k=5 - Error: 0.04

The code is shown in Figure 6.

```

9      model_3 = kMeans(X[y==1,:], k)
10     z_3 = model_3.predict(X[y==1, :])
11     model_5 = kMeans(X[y==2, :], k)
12     z_5 = model_5.predict(X[y==2, :])
13
14     # Compute p(y = 1)
15     p_y = sum(y==1)/n
16
17     # Let's create p_zy
18     p_zy = zeros(k, 2)
19     for c in 1:k
20         p_zy[c, 1] = sum(z_3 == c) / sum(y==1)
21         p_zy[c, 2] = sum(z_5 == c) / sum(y==2)
22     end
23
24     # Let's create p_xyz
25     p_xyz = zeros(d, 2, k)
26     for j in 1:d
27         for c in 1:k
28             p_xyz[j, 1, c] = sum((X[y==1,j]==1) .* (z_3==c))/sum(z_3==c)
29             p_xyz[j, 2, c] = sum((X[y==2,j]==1) .* (z_5==c))/sum(z_5==c)
30         end
31     end
32
33     function predict(Xhat)
34         (t,d) = size(Xhat)
35         yhat = zeros(t)
36
37         for i in 1:t
38             # p_yx = p_y*prod(p_x) for the appropriate x and y values
39             p_yx = [p_y;1-p_y]
40             sum_over_cluster = zeros(2)
41             for c in 1:k
42                 prod_over_features = ones(k, 2)
43                 for j in 1:d
44                     if Xhat[i, j] == 1
45                         prod_over_features[c, 1] *= p_xyz[j, 1, c]
46                         prod_over_features[c, 2] *= p_xyz[j, 2, c]
47                     else
48                         prod_over_features[c, 1] *= 1-p_xyz[j, 1, c]
49                         prod_over_features[c, 2] *= 1-p_xyz[j, 2, c]
50                     end
51                 end
52                 sum_over_cluster[1] = sum_over_cluster[1] + p_zy[c, 1] * prod_over_features[c, 1]
53                 sum_over_cluster[2] = sum_over_cluster[2] + p_zy[c, 2] * prod_over_features[c, 2]
54             end
55             p_yx[1] *= sum_over_cluster[1]
56             p_yx[2] *= sum_over_cluster[2]
57             if p_yx[1] > p_yx[2]
58                 yhat[i] = 1
59             else
60                 yhat[i] = 2
61             end
62         end
63         return yhat
64     end
65
66     return GenericModel(predict)
67 end

```

Figure 6: Vectorized Naive Bayes implementation

2. What is the cost of making a prediction with this model?

Answer: The question is asking for cost of predictions, so we assume we have already computed the θ tables and cost of lookup is $O(1)$. Therefore, for each prediction sample, finding $p(y^i)$ is $O(1)$. If we have k clusters and a dimension of d , finding $\sum_{z=1}^k p(z | y^i) \prod_{j=1}^d p(x_j^i | y^i, z)$ is $O(kd)$. Now, if we have n samples, this becomes $O(nkd)$.

FINAL ANSWER: $O(nkd)$ for n samples and $O(kd)$ for 1 sample

3. For a run of the method with $k = 5$, show the images obtained by plotting the estimates for $p(x_j = 1 | z, y)$ for all j as a 28 by 28 image, for each value of z and y (so there should be 10 images). Hint: if you want to view image i with *PyPlot*, you could use `imshow(reshape(X[i,:],28,28),'gray')`.

Answer: The images are shown in Figure 7. We can see that different clusters within each class correspond to different variations of the same digit.

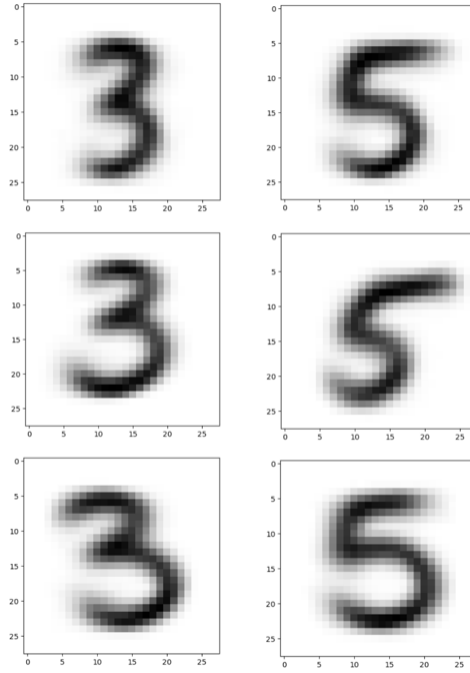


Figure 7: $p(x_j = 1 | z, y)$ visualized

4. The logistic regression model and VQNB model obtain similar accuracies, and both give estimates of the probability for $p(y^i | x_1^i, x_2^i, \dots, x_d^i)$. Give an example of something we could do with the VQNB model that we could not do with the logistic regression model.

Answer: VQNB model is a generative model modelling $p(x, y)$ while logistic regression is a discriminative model modelling $p(y | x)$. Therefore with VQNB, we can find something like $\arg \max_{x_1, \dots, x_d} p(x_1, \dots, x_d | y = c)$ which is conditional decoding. As an example, this can be used to find features that make an email appear like a spam.

3 Neural Networks

3.1 Neural Networks by Hand

Suppose that we train a neural network with sigmoid activations and one hidden layer and obtain the following parameters (assume that we don't use any bias variables):

$$W = \begin{bmatrix} -2 & 2 & -1 \\ 1 & -2 & 0 \end{bmatrix}, v = \begin{bmatrix} 3 \\ 1 \end{bmatrix}.$$

Assuming that we are doing regression, for a training example with features $(x^i)^T = [-3 \ -2 \ 2]$ what are the values in this network of the hidden units z^i , activations $a^i = h(z^i)$, and prediction \hat{y}^i ?

Answer: The formula for a NN with one hidden layer is $\hat{y}^i = v^T h(Wx^i)$. So let's compute each part of this:

- $(z^i)^T = [(-2 * -3) + (2 * -2) + (-1 * 2), (1 * -3) + (-2 * -2) + (0 * 2)] = [0, 1]$
- $(a^i)^T = [\sigma(0), \sigma(1)] = [0.5, 0.731]$
- $\hat{y}^i = 3 * 0.5 + 1 * 0.731 = 2.231$.

3.2 Neural Network Tuning - Regression

The file `example_nnet.jl` runs a stochastic gradient method to train a neural network on the *basisData* dataset from a previous assignment. However, in its current form it doesn't fit the data very well. Modify the training procedure and model to improve the performance of the neural network. [Hand in your plot after changing the code to have better performance, and list the changes you made.](#)

Hint: there are many possible strategies you could take to improve performance. Below are some suggestions, but note that the some will be more effective than others:

- Changing the network structure (*nHidden* is a vector giving the number of hidden units in each layer).
- Changing the training procedure (you can change the stochastic gradient step-size, use decreasing step sizes, use mini-batches, run it for more iterations, add momentum, switch to *findMin*, use Adam, and so on).
- Transform the data by standardizing the features, standardizing the targets, and so on.
- Add regularization (L2-regularization, L1-regularization, dropout, and so on).
- Change the initialization.
- Add bias variables.
- Change the loss function or the non-linearities (right now it uses squared error and tanh to introduce non-linearity).
- Use mini-batches of data, possibly with batch normalization.

Answer: Here are the changes in the order I tried them:

- We added bias to the model by adding a column of 1's to the features matrix: `[ones(n, 1) X]`. This allowed the predicted line to not be constrained to the origin.
- The second (and the most frustrating) improvement I added was support for mini-batch size. The changes I made include changing the `NeuralNet_backprop` function to account for the new shapes when multiplying matrices.
- We used `findMinL1` with `1e-5` for the L1 regularization `lambda` to optimize the objective. Because of the added support for mini-batches, the `NeuralNet_backprop` can be used as the objective function for `findMin`. Additionally, this is possible since `X` can fit onto memory as a whole. `findMin` uses dynamic learning rate so there is no need to add that separately.
- We changed the loss to use Mean Squared Error for the batch; $f = (1/n) * \sum(r.\hat{r})$. This also involved changing the derivative for the loss.
- We changed the network hidden dimensions to `[20, 10, 5, 3]`.
- We increased the number of training epochs to 100000.

The plot before these changes is shown in Figure 8, and the plot after the above changes is shown in Figure 9.

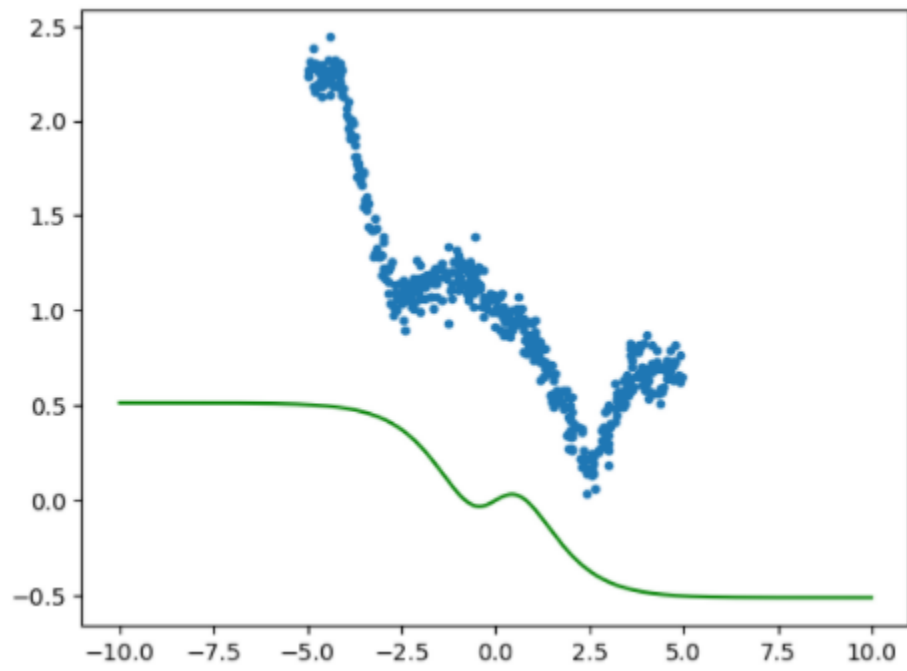


Figure 8: Neural Network performance without improvements

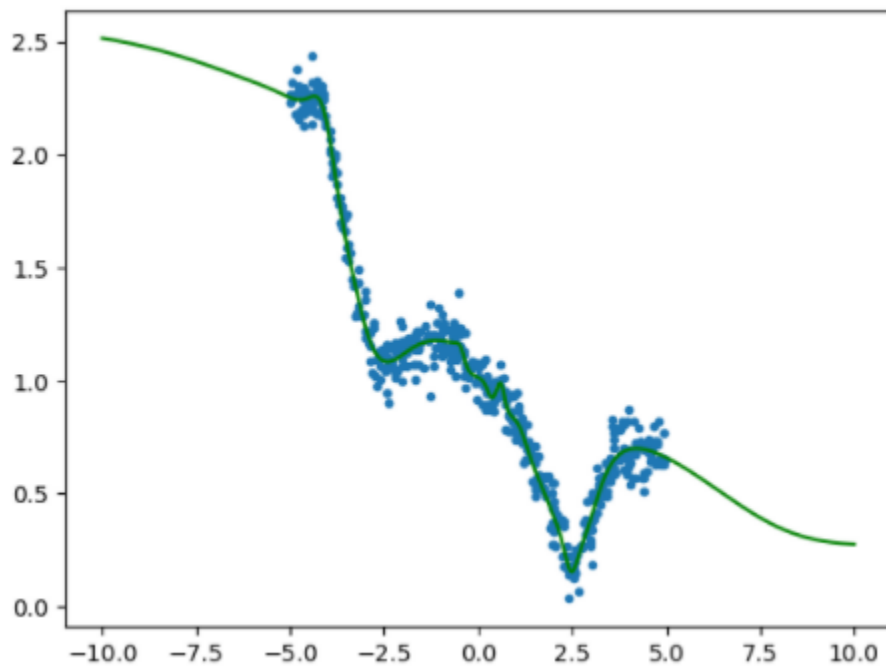


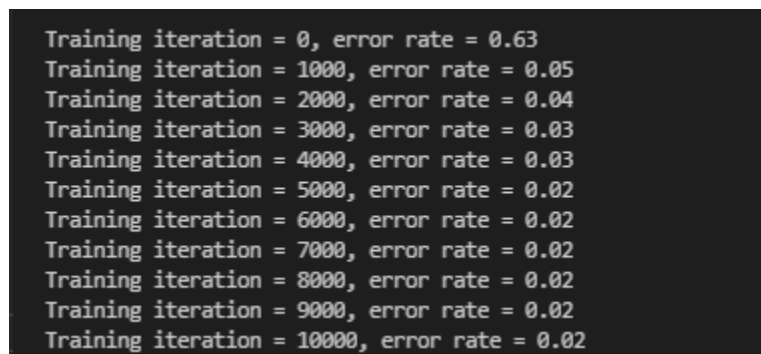
Figure 9: Neural Network performance with improvements

3.3 Neural Network Tuning - Classification

The file `example_mnist35_nnet.jl` runs a stochastic gradient method to train a neural network on the '3' and '5' images from the naive Bayes question (it uses the squared error for training and classifies by taking the sign of the prediction). Modify the training procedure and model so that the neural network has a better error on the test set than the 0.03 achieved by logistic regression. [List the changes you made and the best test performance that you were able to obtain.](#)

Answer: As shown in Figure 10 by making the following changes we get an error rate of 0.02:

- Since the data is higher dimensional here, `findMin` is not the best choice. We added support for mini batches instead and train on batches of size 361.
- We normalized the grey images by dividing pixel values for both train and test images by 255 (more advanced normalization could be done, but not necessary in this case.)
- We changed network hidden dims to `[30, 20, 5]`.
- We added a scheduler for the optimizer step size such that step size is halved when test error plateaus.
- We changed the loss and its associated derivatives to use the Binary Cross Entropy loss
- We added a Sigmoid output activation function
- We added He weight initialization to the model using `sqrt(2.0/size(Wm[layer-1])[2])`



```
Training iteration = 0, error rate = 0.63
Training iteration = 1000, error rate = 0.05
Training iteration = 2000, error rate = 0.04
Training iteration = 3000, error rate = 0.03
Training iteration = 4000, error rate = 0.03
Training iteration = 5000, error rate = 0.02
Training iteration = 6000, error rate = 0.02
Training iteration = 7000, error rate = 0.02
Training iteration = 8000, error rate = 0.02
Training iteration = 9000, error rate = 0.02
Training iteration = 10000, error rate = 0.02
```

Figure 10: Improved classification performance

4 Very-Short Answer Questions

1. List 5 different inference tasks you might do with a (single variable) Bernoulli model?

Answer:

- We can find probability (likelihood) of a single Bernoulli variable (single event): $p(x = 1 \mid \theta)$
- We can find the probability of an IID dataset of Bernoullis: $p(x^1, \dots, x^n \mid \theta)$
- We can decode a single Bernoulli variable by finding x that maximizes $p(x \mid \theta)$
- We can decode an IID dataset of Bernoullis by finding x^i 's that maximize $p(x^1, \dots, x^n \mid \theta)$
- We can generate samples from $p(x \mid \theta)$

2. What is a reason we might want to sample from a distribution?

Answer: It could be used in simulations that involve some randomness. For example, it could be used to simulate a random walk based on some distribution.

3. Suppose we have $p(x) \propto \alpha g(x)$ for some continuous variable x , some positive constant α , and some non-negative function g . Give the form of the distribution if we do not use the \propto sign to hide the proportionality constant.

Answer: $p(x) \propto \alpha g(x)$ can be simplified to $p(x) \propto g(x)$ first. Then $p(x) = kg(x)$ and since $\int_x p(x) = 1$, then $\int_x kg(x) = 1$, then $k = \frac{1}{\int_x g(x)}$. Therefore, final answer is $p(x) = \frac{g(x)}{\int_x g(x)}$.

4. In the CANB model (discussed in a previous question), you would need to estimate $p(x_j^i = 1 \mid x_{j-1}^i, y^i)$ for all values of x_{j-1}^i and y^i . We could consider a generalization of this where we estimate $p(x_j^i = 1 \mid x_{j-1}^i, x_{j-2}^i, \dots, x_{j-k}^i, y^i)$ for some value k (to further relax the conditional independence assumption made by naive Bayes.) Assuming all values are binary we use tabular probabilities, how many parameters would this model need?

Answer: Number of parameter: $d * 2^{k+1}$. Please note that this answer is only for modeling $p(x_j^i = 1 \mid x_{j-1}^i, x_{j-2}^i, \dots, x_{j-k}^i, y^i)$ and not the whole CANB model. If our data is d -dimensional, for each x_j where $j \in d$, we need $2 * 2^k$ parameters where the first two is for y and the second term is for the prior k x_j 's. So in total, we have $d * 2^{k+1}$ parameters.

5. What is the advantage of using a discriminative model for supervised learning, rather than the generative approach of treating supervised learning as special case of density estimation.

Answer: For discriminative models, we don't have to model the features. That is, we don't need to worry about complicated scenarios where features may be correlated and hard to model. We also don't need to model $p(y)$, which is not as big a problem as the last one.

6. Give a choice of the number of hidden units, the activation function, and likelihood in a neural network with a single hidden layer such that MLE in the model makes the same predictions as the MLE logistic regression model.

Answer: For logistic regression, we have the form $\frac{1}{1+\exp(-w^T x^i)}$ where w is a vector of dimension d (feature dimension of x^i). Now, if our Neural Network has the form $\hat{y} = v^T h(Wx)$, we can for a d -dimensional x :

- set h to be the identity operation (or any linear operation)
- W can be an $\hat{d} * d$ matrix where \hat{d} can be anything.
- set v to be a \hat{d} -dimensional vector
- Use sigmoid as the output activation (likelihood) function

The above is true since the main expressive power of NNs comes from their mid-layer non-linear activation functions. By using a linear activation function, our model will be the same as logistic regression.

7. Describe an experimental setup where you might see a double descent curve when fitting a logistic regression model with L2-regularization. You can assume you have a way to generate new relevant features. Caution: the global minimum is unique in this model so the double descent would not come from choosing among multiple global minima.

Answer: We know that increasing the capacity of the model can lead to the double gradient descent scenario. So, we can train the model on the initial given features, and when the model overfits, we add generated relevant features and allow the model to fit the new features which could lead to test performance improvement. (Credits: with help from course TA and Xinyu Kang)

8. Consider a neural network with L hidden layers, where each layer has at most k hidden units and we have c labels in the final layer (and the number of features is d). What is the cost of prediction with this network?

Answer: We know that the first layer matrix multiplication would have $O(kd)$. The subsequent layers (excluding the last one) have $O(k^2)$, and the last layer has $O(ck)$. Additionally, the activation functions for each layer would be $O(k)$, which dominated by other terms. Therefore, the final answer is $O(O(kd) + O(ck) + (L - 1)k^2)$

9. What is one advantage of using automatic differentiation and one disadvantage?

Answer: One advantage is that the cost of getting gradients is the same as cost of computing the function. However, this could also a misadvantage in some cases where simplistic closed-form formulas can be obtained for the function. Also it has to store all intermediate calculations, which is not very storage-efficient.

10. Convolutional networks seem like a pain... why not just use regular (“fully connected”) neural networks for image classification?

Answer: For two main reasons:

- For fully connected NNs, we have to flatten the images, which causes spatial information to be lost.
- The number of parameters for FC NNs gets huge very quickly with image size, whereas in CNNs the filter size can remain constant.

11. Why do autoencoders have a bottleneck layer?

Answer: The bottleneck layer maps data into a lower-dimensional space, which could be seen as a feature selection and feature reduction operation. The decoder can be removed after pre-training so the embeddings from the bottleneck layer can be used in other downstream tasks or visualizations.

12. We discussed multi-label classification using neural networks and a product of Bernoullis model. The product of Bernoullis model assumes the class labels are independent, so why might this model make predictions that seem to capture dependencies between the random variables?

Answer: This model only assumes independence among labels conditioned on the last layer of the Neural Network. That is the dependency is already captured by the NN in its last layer.

13. Why can fully-convolutional networks segment images of different sizes?

Answer: We are using convolutional filters so the same parameters are used across space at all layers. In other words, we can convolve a filter on images of different sizes.

CPSC 440/540 Machine Learning (January-April, 2022)

Assignment 3 (due Friday March 11th at midnight)

For this assignment you can work in groups of 1-2. However, please only hand in one assignment for the group.

1. Name(s):

Answer: Masoud Mokhtari, Ali Mehrabian

2. Student ID(s):

Answer: 14186167, 31662323

1 Bayesian Learning

1.1 Conjugate Priors

Consider counts $y \in \{0, 1, 2, 3, \dots\}$ following a Poisson distribution with rate parameter $\lambda > 0$,

$$p(y \mid \lambda) = \frac{\lambda^y \exp(-\lambda)}{y!}.$$

We'll assume that λ follows a Gamma distribution (the conjugate prior to the Poisson) with shape parameter $\alpha > 0$ and rate parameter $\beta > 0$,

$$\lambda \sim \text{Gamma}(\alpha, \beta),$$

or equivalently that

$$p(\lambda \mid \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} \exp(-\beta\lambda),$$

where Γ is the gamma function.

Hint: You should be able to use the form of the gamma distribution to solve all the integrals that show up in this question. You can use $Z(\alpha, \beta) = \frac{\Gamma(\alpha)}{\beta^\alpha}$ to represent the normalizing constant of the gamma distribution, and use α^+ and β^+ as the updated parameters of the gamma distribution in the posterior. Compute the following quantites:

1. The posterior distribution,

$$p(\lambda \mid y, \alpha, \beta).$$

$$\text{Answer: } p(\lambda \mid y, \alpha, \beta) = \frac{p(y_0, \dots, y_{n-1} \mid \lambda) p(\lambda \mid \alpha, \beta)}{\int p(y_0, \dots, y_{n-1} \mid \lambda') p(\lambda' \mid \alpha, \beta) d\lambda'} = \frac{\frac{\beta^\alpha}{\prod y!} \frac{\lambda^{\alpha+\sum y-1}}{\Gamma(\alpha)} \exp(-(\beta+n)\lambda)}{\int \frac{\beta^\alpha}{\prod y!} \frac{\lambda'^{\alpha+\sum y-1}}{\Gamma(\alpha)} \exp(-(\beta+n)\lambda') d\lambda'}$$

$$\frac{\lambda^{\alpha+\sum y-1} \exp(-(\beta+n)\lambda)}{\int \lambda'^{\alpha+\sum y-1} \exp(-(\beta+n)\lambda') d\lambda'} = \frac{\frac{\lambda^{\alpha+\sum y-1} \exp(-(\beta+n)\lambda)}{Z(\alpha+\sum y, \beta+n)}}{\underbrace{\int \frac{\lambda'^{\alpha+\sum y-1} \exp(-(\beta+n)\lambda')}{Z(\alpha+\sum y, \beta+n)} d\lambda'}_{\text{Gamma}(\alpha+\sum y, \beta+n) \text{ pdf}=1}} = \frac{\lambda^{\alpha+\sum y-1} \exp(-(\beta+n)\lambda)}{Z(\alpha+\sum y, \beta+n)} =$$

$$\frac{(\beta+n)^{\alpha+\sum y}}{\Gamma(\alpha+\sum y)} \lambda^{\alpha+\sum y-1} \exp(-(\beta+n)\lambda) \longrightarrow p(\lambda \mid y, \alpha, \beta) \sim \text{Gamma}(\alpha+\sum y, \beta+n)$$

The posterior follows a Gamma distribution with new parameters $\alpha^+ = \alpha + \sum y, \beta^+ = \beta + n$.

In the case that we only consider a single y , the parameters would be $\alpha^+ = \alpha + y, \beta^+ = \beta + 1$.

This applies to the rest of the questions.

2. The marginal likelihood of y given the hyper-parameters α and β ,

$$p(y \mid \alpha, \beta) = \int p(y, \lambda \mid \alpha, \beta) d\lambda.$$

$$\text{Answer: } p(y \mid \alpha, \beta) = \int p(\lambda \mid y, \alpha, \beta) p(Y \mid \lambda) d\lambda = \int \frac{(\beta^+)^{\alpha^+}}{\Gamma(\alpha^+)} \lambda^{\alpha^+-1} \exp(-\beta^+ \lambda) \frac{\lambda^{\sum y} \exp(-n\lambda)}{\prod y!} d\lambda$$

$$\frac{(\beta^+)^{\alpha^+}}{(\prod_i y_i!) \Gamma(\alpha^+)} Z(\alpha^+ + \sum y, \beta^+ + n) \underbrace{\int \frac{\lambda^{\alpha^++\sum y-1} \exp(-(\beta^+ + n)\lambda)}{Z(\alpha^+ + \sum y, \beta^+ + n)} d\lambda}_{\text{Gamma}(\alpha^++\sum y, \beta^++n) \text{ pdf}=1}$$

$$= \frac{Z(\alpha^+ + \sum y, \beta^+ + n)}{Z(\alpha^+, \beta^+)} \times \frac{1}{\prod_i y_i!}$$

In the case that we only consider a single y , the final answer would be $\frac{Z(\alpha^+ + y, \beta^+ + 1)}{Z(\alpha^+, \beta^+)} \times \frac{1}{y!}$ where $\alpha^+ = \alpha + y, \beta^+ = \beta + 1$.

3. The posterior mean estimate for λ ,

$$\mathbb{E}_{\lambda \mid y, \alpha, \beta}[\lambda] = \int \lambda p(\lambda \mid y, \alpha, \beta) d\lambda.$$

$$\text{Answer: } \mathbb{E}_{\lambda \mid y, \alpha, \beta}[\lambda] = \int \frac{(\beta^+)^{\alpha^+}}{\Gamma(\alpha^+)} \lambda^{\alpha^+} \exp(-\beta^+ \lambda) d\lambda = \frac{(\beta^+)^{\alpha^+}}{\Gamma(\alpha^+)} \int \lambda^{\alpha^+} \exp(-\beta^+ \lambda) d\lambda$$

$$= \frac{(\beta^+)^{\alpha^+}}{\Gamma(\alpha^+)} \frac{\Gamma(\alpha^+ + 1)}{(\beta^+)^{\alpha^++1}} = \frac{(\beta^+)^{\alpha^+}}{\Gamma(\alpha^+)} \frac{\alpha^+ \Gamma(\alpha^+)}{(\beta^+)^{\alpha^++1}} = \frac{\alpha^+}{\beta^+}$$

We used the gamma function property $\Gamma(\alpha + 1) = \alpha \Gamma(\alpha)$

4. The posterior predictive distribution for a new independent observation \tilde{y} given y ,

$$p(\tilde{y} \mid y, \alpha, \beta) = \int p(\tilde{y}, \lambda \mid y, \alpha, \beta) d\lambda.$$

$$\text{Answer: } p(\tilde{y} \mid y, \alpha, \beta) = \int \frac{(\beta^+)^{\alpha^+}}{\Gamma(\alpha^+)} \lambda^{\alpha^+ - 1} \exp(-\beta^+ \lambda) \frac{\lambda^{\tilde{y}} \exp(-\lambda)}{\tilde{y}!} d\lambda \xrightarrow{\text{similar to 2}}$$

$$\frac{(\beta^+)^{\alpha^+}}{\tilde{y}! \Gamma(\alpha^+)} Z(\alpha^+ + \tilde{y}, \beta^+ + 1) \underbrace{\int \frac{\lambda^{\alpha^+ + \tilde{y} - 1} \exp(-(\beta^+ + 1)\lambda)}{Z(\alpha^+ + \tilde{y}, \beta^+ + 1)} d\lambda}_{\text{Gamma}(\alpha^+ + \tilde{y}, \beta^+ + 1) \text{ pdf} = 1} = \frac{Z(\alpha^+ + \tilde{y}, \beta^+ + 1)}{Z(\alpha^+, \beta^+)} \times \frac{1}{\tilde{y}!}$$

1.2 Empirical Bayes

If you run the script `example_groups.jl` it will load a dataset representing the number of people diagnosed with stomach cancer in a set of different regions, and the number of people at risk in the region. The format of the data is as follows:

- `n[j,1]` is number of positive examples in the training set for group j .
- `n[j,2]` is the total number of examples in the training set for group j .
- `nTest[j,1]` is number of positive examples in the testing set for group j .
- `nTest[j,1]` is the total number of examples in the testing set for group j .

The code first shows the negative log-likelihood (NLL) on the test set if we assume that the probability of getting stomach cancer among each group is 0.5. The NLL under this choice is very high, since the number of positives is fairly low (remember that we want to have a low NLL, corresponding to a high likelihood). The code then computes the MLE for each group, and evaluates the NLL with this choice.

1. Why do we get NaN for the NLL with the MLE values? What is the actual test likelihood supposed to be in this case?

Answer: Inspecting the training dataset, we notice that many groups have no positive examples. Computing the MLE as $\hat{\theta}_j = \frac{n_{1j}}{n_j}$ would cause $\hat{\theta}_j$ to be 0 when the number of positive samples is 0 in group j . When the number of positive samples in the test set is also 0, the $\log(\hat{\theta}_j)$ term in the NLL makes the results undefined. In reality, we expect θ for groups where no positive samples are observed in the dataset to be a small non-zero value as there still is a chance of seeing cancer given other factors.

For the groups where n_1 for both training and test set are 0, $p(X_{\text{test}j}|\hat{\theta}_j = 0) = 0^0 * 1^{n_{0j}} = 1$. However, if n_1 is 0 in the training set and non-zero in the test set, we get $p(X_{\text{test}j}|\hat{\theta}_j = 0) = 0^{n_1} * 1^{n_0} = 0$. We can use the code shown in Figure 1 and see that the **overall test likelihood is 0**.

```
19 # Function to compute NLL when we have a theta for each group
20 LLs(theta,n) =
21 begin
22     LL = 1
23     for j in 1:k
24         LL *= theta[j]^n[j,1] * (1-theta[j])^(n[j,2]-n[j,1])
25     end
26     return LL
27 end
28
29 # Fit MLE for each training group
30 MLEs = n[:,1]./n[:,2]
31 @show LLs(MLEs,nTest)
32
```

Figure 1: Test likelihood code for MLE.

2. Laplace smoothing corresponds to MAP estimation with a beta prior that has $\alpha = 2$ and $\beta = 2$. What is the test NLL if we use Laplace smoothing to estimate the parameters?

Answer: We know that for a Bernoulli-Beta model, MAP is computed as $\hat{\theta}_j = \frac{n_{pj} + \alpha - 1}{n_j + \alpha + \beta - 2}$. Therefore by changing the code to what is shown in Figure 2, we get an **NLL of 274.68**.

```

 9  # Function to compute NLL when we have a theta for each group
10  NLLs(theta,n) =
11  begin
12      LL = 0
13      for j in 1:k
14          LL -= n[j,1]*log(theta[j]) + (n[j,2]-n[j,1])*log(1-theta[j])
15      end
16      return LL
17  end
18
19  # Fit MAP for each training group
20  a = 2
21  b = 2
22  MAPs = (n[:,1].+a.-1)./(n[:,2].+a.+b.-2)
23
24  # Show training and test NLL
25  @show NLLs(MAPs,nTest)
26

```

Figure 2: NLL with MAP estimation

3. Instead of using MAP estimation and then computing the NLL, we could use the Bayesian approach of computing the negative logarithm of the posterior predictive probability of the test data. [What is the negative log of the posterior predictive probability of the test data under a beta prior with \$\alpha = 2\$ and \$\beta = 2\$? Explain why you think this gives better/worse results than the NLL under MAP estimation.](#)

Answer: The posterior predictive is (\hat{n} is for the test set and n is for the training set):

$$\begin{aligned}
 p(X_{j\text{test}}|X_{j\text{train}}, \alpha, \beta) &= \frac{1}{B(\alpha + n_{1j}, \beta + n_{0j})} \int \theta_j^{\hat{n}_{1j}} (1 - \theta_j)^{\hat{n}_{0j}} \theta_j^{\alpha + n_{1j} - 1} (1 - \theta_j)^{\beta + n_{0j} - 1} d\theta_j \\
 &= \frac{1}{B(\alpha + n_{1j}, \beta + n_{0j})} \int \theta_j^{\alpha + n_{1j} + \hat{n}_{1j} - 1} (1 - \theta_j)^{\beta + n_{0j} + \hat{n}_{0j} - 1} d\theta_j \\
 &= \frac{B(\alpha + n_{1j} + \hat{n}_{1j}, \beta + n_{0j} + \hat{n}_{0j})}{B(\alpha + n_{1j}, \beta + n_{0j})}
 \end{aligned}$$

Therefore negative log of posterior is:

$$-\log(B(\alpha + n_{1j} + \hat{n}_{1j}, \beta + n_{0j} + \hat{n}_{0j})) + \log(B(\alpha + n_{1j}, \beta + n_{0j}))$$

Using the code shown in Figure 3, the **negative log of posterior is 280.46**. Although we may expect the Bayesian model to have better performance due to trusting the data less and performing some type of averaging and regularization, we see the opposite here. We hypothesize that this is due to $\alpha = 2$ and $\beta = 2$ not being good prior parameters for a case where the number of positive samples is a lot less than negative ones. To prove this, using $\alpha = 0.1$ and $\beta = 2$, the Bayesian model yields a log likelihood of 265.8, which is a lot better. Additionally, MAP is doing better with this bad prior since it has a higher reliance on data compared to Bayesian.

```

1 using JLD
2 using SpecialFunctions
3 data = load("cancerData.jld")
4 n = data["n"]
5 nTest = data["nTest"]
6
7 # Compute number of groups (here, the number of training and testing groups is the same)
8 k = size(n,1)
9
10 # Function to compute NLPP when we have a theta for each group
11 NLPP(n_test, n_train, a, b) =
12 begin
13     LL = 0
14     for j in 1:k
15         LL -= log(beta(a+n_train[j, 1]+n_test[j, 1], b + (n_train[j,2] - n_train[j, 1]) + (n_test[j, 2] - n_test[j, 1])))
16         LL += log(beta(a+n_train[j, 1], b+(n_train[j, 2] - n_train[j, 1])))
17     end
18     return LL
19 end
20
21 # Show training and test NLL for MLE
22 @show NLPP(nTest, n, 2, 2)
23 | 280.4598944058339

```

Figure 3: Bayesian inference for Bernoulli-Beta using Monte Carlo

4. Instead of modeling each group independently, consider fitting a single Bernoulli model by pooling the data across all groups. In other words, we ignore the groups and treat all the examples as if they came from a single source. Report the test NLL in this pooled setting when using MLE, MAP, and the posterior predictive. Why are these results more similar than when we use independent Bernoullis for each group?

Answer: Using the code in Figure 4, we get the following results

- MLE: 263.9
- MAP: 264.0
- Bayesian: 264.2

```

7 n_total_train = 0
8 n_pos_train = 0
9 n_total_test = 0
10 n_pos_test = 0
11
12 # collate samples
13 for i in 1:size(n, 1)
14     n_total_train += n[i,2]
15     n_pos_train += n[i,1]
16 end
17 for i in 1:size(nTest, 1)
18     n_total_test += nTest[i,2]
19     n_pos_test += nTest[i,1]
20 end
21
22 # Function to compute NLL
23 NLLs(theta,n_pos,n_total) =
24 begin
25     LL = -(n_pos*log(theta) + (n_total-n_pos)*log(1-theta))
26 end
27
28 # Function to compute NLPP
29 NLPP(n_pos_test,n_total_test,n_pos_train,n_total_train,a,b) =
30 begin
31     LL = -log(beta(a+n_pos_train+n_pos_test,b+(n_total_train-n_pos_train)+(n_total_test-n_pos_test)))
32     LL += log(beta(a+n_pos_train,b+(n_total_train-n_pos_train)))
33 end
34
35 # Show test NLL if all theta=0.5
36 theta = 0.5
37 @show NLLs(theta,n_pos_test,n_total_test)
38
39 # Fit MLE for each training group
40 MLE = n_pos_train/n_total_train
41
42 # Show training and test NLL for MLE
43 @show NLLs(MLE,n_pos_test,n_total_test)
44
45 # Fit MAP for each training group
46 a = 2
47 b = 2
48 MAP = (n_pos_train+a-1)/(n_total_train+a+b-2)
49
50 # Show training and test NLL for MLE
51 @show NLLs(MAP,n_pos_test,n_total_test)
52
53 # Show training and test NLL for MLE
54 @show NLPP(n_pos_test,n_total_test,n_pos_train,n_total_train,a,b)
55
56 | 264.20196506732566

```

Figure 4: Pooled dataset estimation implementation

We see that all methods are converging to similar answers. This is because by pooling groups together, it is like we have a larger dataset with more samples. With a high number of samples, all these estimations get closer to the true likelihood and perform similarly.

5. In the pooled data model, you can compute the logarithm of the marginal likelihood of the hyperparameters given the data as $\text{logMargLik}(a,b) = \text{logbeta}(a+n_1,b+n_0) - \text{logbeta}(a,b)$ (which is the ratio of posterior and prior normalizing constants, converted to the log domain).¹ Here, n_1 is the number of 1s, n_0 is the number of 0s, a gives the value of α , and b gives the value of β . [Can you find the value of \$\alpha\$ and \$\beta\$ that optimize the marginal likelihood?](#) Hint: it may be helpful to parameterize the beta distribution in terms of $m = \alpha/(\alpha + \beta)$ (the proportion of 1s) and $k = (\alpha + \beta)$ (the “strength” of our belief in this ratio). Try setting m to the MLE value of θ and varying k .

Answer: Let’s do the reparametrization first: $\alpha = mk$ and $\beta = k - mk = k(1 - m)$. Therefore, we have:

$$\text{logMargLike}(m,k) = \text{logbeta}(mk + n_1, k(1 - m) + n_0) - \text{logbeta}(mk, k(1 - m))$$

Setting m to $\frac{n_1}{n_1+n_0}$, and using the code shown in Figure 5, we see that increasing k keeps improving the likelihood, and it seems not possible to find the optimal value of α and β .

¹This uses the “SpecialFunctions” package to implement `logbeta`, the logarithm of the beta function (the beta function is the normalizing constant of the beta distribution).

```

11 # collate samples
12 k = size(n, 1)
13 for i in 1:k
14     n_total_train += n[i,2]
15     n_one_train += n[i,1]
16 end
17 n_zero_train = n_total_train - n_one_train
18
19 # Fit MLE for each training group
20 MLE = n_one_train/n_total_train
21
22 best_k = -999999
23 best_ll = -999999
24 for k in 0:10:100000
25     ll = logbeta(MLE*k+n_one_train, k*(1-MLE)+n_zero_train) - logbeta(MLE*k, k*(1-MLE))
26     if ll > best_ll
27         best_k = k
28         best_ll = ll
29     end
30 end

```

Figure 5: Finding best k - unsuccessful

6. In the textbook where I got this data,² they suggest using an independent hyper-prior over m and k of the form

$$p(m) \propto m^{0.1-1}(1-m)^{9.9-1}, \quad p(k) \propto 1/(1+k)^2.$$

The hyper-prior over m is biased towards low values (since we expect the cancer to be rare) but not particularly strong, while the prior over k is similar to what is called a “Jeffreys prior” over scale variables (which would satisfies a particularly definition of being an “uninformative prior”). In the pooled data model, find the value of α and β that optimize the marginal likelihood with this hyper-prior (or equivalently optimize the log of the marginal likelihood with the log of this prior as the regularizer). Up to one decimal place, what are the optimal values of α and β under this hyper-prior? What values of m and k does this correspond to choosing? Hand in your code for computing the objective function that is being optimized. Hint: for this question you can use a “brute force” approach to find α and β , by searching over all values of the form “x.y” for x in 0-9 and y in 0-9 (but where at least one of “x” and “y” must be bigger than 0).

Answer: Using the code shown in Figure 6, we find the best α to be 0.1 and the best β to be 2.6. These correspond to $m = 0.037$ and $k = 2.7$.

```

19 valid_values = []
20 for x in range(start=0, stop=9, step=1)
21     for y in range(start=0, stop=0.9, step=0.1)
22         if x+y > 0
23             append!(valid_values, x+y)
24         end
25     end
26 end
27
28 best_LL = -99999
29 best_a = -10
30 best_b = -10
31 for a in valid_values
32     for b in valid_values
33         m = a/(a+b)
34         k = a+b
35         LL = logbeta(n_one_train+a, n_zero_train+b) - logbeta(a,b) - 0.99*log(m) + 8.9*log(1-m) - 2*log(1+k)
36         if LL > best_LL
37             best_LL = LL
38             best_b = b
39             best_a = a
40         end
41     end
42 end

```

Figure 6: Finding the best prior hyper-parameters for collated dataset.

²“Ordinal Data Modeling” by Johnson and Albert.

7. We can also use the (regularized) marginal log-likelihood to learn the hyper-parameters in our original setting where we had a separate parameter for each group. In this setting the marginal likelihood is given by

$$p(X | \alpha, \beta) \propto \prod_{j=1}^k \frac{Z(\alpha + n_{1j}, \beta + n_{0j})}{Z(\alpha, \beta)},$$

where n_{1j} is the number of 1s in group j , n_{0j} is the number of 0s in group j , and the normalizing constant Z is the beta function. Find the values of α and β that optimize the marginal likelihood multiplied by the hyper-prior from the previous question, up to 1 decimal place (the value “x” in the brute-force search will need to be increased). [Hand in your code to compute the objective function being optimized, report the values of \$\alpha\$ and \$\beta\$ that you find, and report the posterior predictive probability of the test data under this choice of \$\alpha\$ and \$\beta\$.](#)

Answer: Using the code shown in Figure 7, we find $\alpha = 1$ and $\beta = 731$ to optimize our objective function. Using a similar code to what is shown in Figure 3, we find the predictive posterior of the test set to be 263.84.

```

20 best_LL = -99999
21 best_a = -10
22 best_b = -10
23 best_m = -10
24 best_k = -10
25 for a in range(1, 1000)
26     for b in range(1, 1000)
27         m = a/(a+b)
28         k = a+b
29         LL = 0
30         for group in 1:num_groups
31             n_one = n[group,1]
32             n_zero = n[group,2] - n[group,1]
33             LL += logbeta(n_one+a, n_zero+b) - logbeta(a,b)
34         end
35         LL += -0.99*log(m) + 8.9*log(1-m) - 2*log(1+k)
36         if LL > best_LL
37             best_LL = LL
38             best_b = b
39             best_a = a
40             best_m = m
41             best_k = k
42         end
43     end
44 end
45

```

Figure 7: Finding the best hyper-parameters for the original datasets.

8. In the model from the last question, [what is the posterior predictive probability of seeing a 1 for a new group?](#)

Answer:

$$p(x = 1 | X, \alpha, \beta) = \int \theta \text{Beta}(\alpha, \beta) d\theta = \frac{\alpha}{\alpha + \beta}$$

The above is using the expectation value of θ under a Beta distribution. Additionally, we just use α and β since $n_1 = 0$ and $n_0 = 0$ for a new group. The final answer is 0.001366, which is the value of m .

9. Under the choice of α and β from the previous question, what is the NLL of the test data if we use MAP estimation for the parameters?

Answer: Using the same code shown in Figure 2 but with different α and β values, we get NaN as the answer. The reason for this is similar to part 1 because when $n_{1j} = 0$ the MAP estimate becomes 0 with our choice of α and β , causing the log likelihood to become NaN for cases where no positive samples are in the test set as well (due to $0 \times \infty$).

10. In this question, many of the best-performing methods achieve similar performance. Give an explanation for why these different model choices do not make a big difference for this dataset.

Answer: Two reasons could be contributing to this. Firstly, we have a simplistic binary dataset, and we use a choice of distributions that allow us to take advantage of conjugate priors. Due to the simplistic nature of this setting, all methods perform similarly, and more involved model choices do not show much improvements compared to more simpler models. Secondly, the size of data (as we observed before) causes all methods to perform similarly. For our pooled model, this is already discussed. In the hyper-prior case with large k , we are indirectly providing a larger number of samples by incorporating that information in our hyper-prior.

2 More Deep Learning

The script `example_flux.jl` shows how to compute the loss function and gradient of a neural network with one hidden layer (with 3 units) using a variety of strategies, and how to update the parameters based on an SGD update using a variety of strategies. This includes the backpropagation code from the previous assignment, as well as automatic differentiation and other functionality provided by the `Flux` package. The function does not produce any output, but you can verify that all methods return the same results up to numerical precision.

1. On the previous assignment we concatenated all parameters into one big vector of parameters w . What is an advantage of the other approaches in the script, where we use a matrix W and a vector v ?

Answer: One thing I had difficulty with for the previous assignment was initializing the model weights using methods such as He initialization, where the weights depend on the number of inputs to a layer. If we have separate weights for each layer, we can properly initialize them without having to go through tedious and error-prone reshaping operations. Additionally, being able to separately observe the content of W and v , we can inspect model's behaviour, which can help in debugging.

2. If we use `Dense(W)` for the first layer in the network, how many parameters does the first layer have and what do the extra parameters represent? Hint: you can use `params(layer)` to get the parameters of a layer.

Answer: We observe that `Dense(W)` contains two sets of parameters, one having a dimension of (3, 784) and another having a dimension of (3,). The first set contains the parameters of our hidden layer that are multiplied by the input matrix. The second set are the bias parameters for the 3 hidden units. In total, we have 2355 parameters.

3. Give code for implementing a neural network with 2 hidden layers, using the `Dense` function within the `Chain` function.

Answer: The code is shown in Figure 8

```
58 # Two-layer Dense model
59 W1 = randn(128,d)
60 W2 = randn(3, 128)
61 vt = reshape(v,1,3)
62 model3 = Chain(Dense(W1, true, relu), Dense(W2, true, relu), Dense(vt, false, identity))
63 loss4(x,y) = (1/2)*(model3(x)[1]-y)^2
64 f_layer = loss4(X[i,:],y[i])
65 g_layer = gradient(params(model3)) do
66     loss4(X[i,:],y[i])
67 end
68 gw1_layer = g_layer[W1]
69 gw2_layer = g_layer[W2]
70 gv_layer = g_layer[vt]
71
```

Figure 8: Creating a Two-Layer Dense Model with Flux.

4. In one of the Flux tutorials, they use the `softmax` function as the last argument to the `Chain` function (for multi-class classification). What is a reason we might not want to include the `softmax` function within the `Chain` function?

Answer: If the distribution of a certain class is a lot higher than all the other ones, we get close to the tails of the softmax function where the gradient is close to 0. This can halt training due to gradients collapsing to 0, which is also known as vanishing gradients.

5. It we use `Conv((5,5),3=>6,relu)` for a layer, [why do we get a layer with 456 parameters?](#)

Answer: We have 6 filters with each one having $5 * 5 * 3$ number of parameters. So before bias, we have a total of $6 * 5 * 5 * 3 = 450$ parameters. Having a single bias parameters for each filter, we get 456 parameters.

6. Re-write the model and training procedure you developed for Assignment 2 Question 3.3 to use one of these alternate ways to compute the gradient and the update of the parameters (so you should not be calling `NeuralNet.jl` anymore). [Re-state the changes you made, hand in your modified code, and report whether you noticed any performance differences \(in terms of run-time, memory, or accuracy\).](#)

Answer: In the list below, we summarize the changes we made for assignment 2 and how we adapted our code to use Flux for this assignment:

- We added support for mini batches and trained on batches of size 361 - No changes for Flux
- We standardized the images by subtracting the mean pixel value and dividing by the standard deviation of pixel values - No changes for Flux
- We changed network hidden dimensions to [30, 20, 5] - We use Flux's Chain and Dense functions to create this 3 hidden-layer network.
- We added a scheduler for the optimizer to halve the step size when progress plateaus - No changes for Flux
- We used a binary cross entropy loss - We simply use Flux's `binarycrossentropy` loss instead of manually creating the loss function
- We added Sigmoid output activation function - For Flux, we change the activation function for the last Dense layer to Sigmoid
- We added He weight initialization - For Flux, we used the `init` option of Dense function with `Flux.kaiming_uniform`.

Our modified code is shown in Figure 9.

```

11 data_std = std(X)
12 data_mean = mean(X)
13
14 X = X.-data_mean ./ data_std
15 Xtest = Xtest.-data_mean ./ data_std
16
17 # Model definition using flux
18 model = Chain(Dense(d, 30, tanh, init=Flux.kaiming_uniform),
19               Dense(30, 20, tanh, init=Flux.kaiming_uniform),
20               Dense(20, 5, tanh, init=Flux.kaiming_uniform),
21               Dense(5, 1, sigmoid, init=Flux.kaiming_uniform))
22 loss(x, y) = Flux.binarycrossentropy(model(x), y)
23 ps = params(model)
24
25 # Define the optimizer
26 stepSize = 1e-2
27 opt = Descent(stepSize)
28
29 # Train with stochastic gradient
30 maxIter = 50000
31 samples_per_batch = 361
32 patience = 3
33 prev_err = 9999
34 for t in 1:maxIter
35
36     i = rand(1:n, samples_per_batch)
37
38     grads = Flux.gradient(ps) do
39         loss(transpose(X[i, :]), reshape(y[i], 1, size(y[i], 1)))
40     end
41     update!(opt, ps, grads)
42
43     # Every few iterations, plot the data/model:
44     if (mod(t-1,round(1000)) == 0)
45         yhat = model(transpose(Xtest)) .> 0.5
46         err = sum(yhat .!= reshape(ytest, 1, size(ytest, 1)))/size(Xtest,1)
47         @printf("Training iteration = %d, error rate = %.2f\n",t-1,err)
48
49         if err < prev_err
50             patience = 3
51             prev_err = err
52         else
53             if patience == 1
54                 stepSize /= 10
55                 opt = Descent(stepSize)
56                 @printf("Reduced step size to %.4f", stepSize)
57                 patience = 3
58             else
59                 patience -= 1
60             end
61         end
62     end
63 end

```

Figure 9: Flux implementation of our network.

Comparisons: As shown in Figure 10, both models achieve very similar error rate.

NeuralNet	Flux
Training iteration = 0, error rate = 0.49	Training iteration = 0, error rate = 0.42
Training iteration = 1000, error rate = 0.02	Training iteration = 1000, error rate = 0.04
Training iteration = 2000, error rate = 0.02	Training iteration = 2000, error rate = 0.03
Training iteration = 3000, error rate = 0.02	Training iteration = 3000, error rate = 0.02
Training iteration = 4000, error rate = 0.02	Training iteration = 4000, error rate = 0.02
Training iteration = 5000, error rate = 0.02	Training iteration = 5000, error rate = 0.02
Training iteration = 6000, error rate = 0.02	Training iteration = 6000, error rate = 0.01
Training iteration = 7000, error rate = 0.01	Training iteration = 7000, error rate = 0.01

Figure 10: Error rate comparison between Flux and NeuralNet implementations.

Since Flux uses auto differentiation and has to save intermediate values, it has higher memory consumption compared to our NeuralNet implementation. Lastly, since automatic differentiation's backpropagation has the same computation complexity as its forward path, Flux's implementation is slower than our NeuralNet implementation. However, it must be noted that this is because Flux's implementation is more general and allows for many different network structures, whereas our NeuralNet implementation is more specialized.

3 Very-Short Answer Questions

1. When we use categorical variables, we assume that the categories do not have any special ordering. But when we sample from a categorical distribution, we use the CDF $p(x \leq c)$. Why does it make sense to use the CDF for sampling if the categories are unordered?

Answer: Because by using the CDF, we divide the unit length into regions depending on the probability of each class. A sample from the uniform distribution (in $[0, 1]$) would fall within one of these regions and the chance of falling within this region is only based on the probability of the corresponding category and not its order.

2. Monte Carlo methods approximate the expectation of a function of a random variable, but we said that they can be used to approximate probabilities. What random function would you use to approximate the probability of an event?

Answer: Probability of an event can be approximated by generating samples from the model and computing the fraction of samples where the condition for that event holds. More formally, using a random function defined as $g(x) = I_{\text{event}}(x)$ where I_{event} is the indicator function for the event of interest, the probability of the event can be approximated as $p(\text{event}) \approx \frac{1}{N} \sum_{i=1}^N g(x^i)$.

3. For the categorical likelihood, what is the relationship between the usual parameters θ_c and the parameters $\hat{\theta}_c$ we used to derive the MLE.

Answer: The parameters $\hat{\theta}$ used for MLE are unconstrained (unnormalized), and therefore, we have the following relationship:

$$\theta_c = \frac{\hat{\theta}_c}{\sum_{i=1}^K \hat{\theta}_c}$$

4. How do MAP and Bayesian methods differ in how make predictions?

Answer: MAP makes predictions based only on the $\tilde{\theta}_c$ with highest posterior, whereas Bayesian methods consider all possible values of θ and weight them according to posterior probability to make predictions.

5. Why do we say that Bayesian methods incorporate regularization, even though there is no regularization term in the posterior predictive distribution.

Answer: Bayesian methods use the average over models weighted by posterior. This posterior includes a prior, and we know that the negative log of the prior correspond to regularization. Therefore, Bayesian models include implicit regularization through the use of priors and model averaging.

6. What is the difference the standard MLE and type II MLE?

Answer: In the standard MLE, we optimize the likelihood of data given parameters θ : $\text{argmax}_{\theta} \{p(X|\theta)\}$. Whereas, for Type II MLE, we optimize the marginal likelihood of data given hyper-parameters by integrating the parameters out: $\text{argmax}_{\alpha} \{p(X|\alpha)\} = \text{argmax}_{\alpha} \{\int p(X|\theta)p(\theta|\alpha)d\theta\}$

7. How does using a conjugate prior simplify the marginal likelihood calculation?

Answer: Using conjugate prior simplifies marginal likelihood calculations because it results in a closed-form formula. That is, the terms inside the integral create an unnormalized distribution similar to the prior, making the integral equal to the normalization factor of the posterior distribution. So, the final formula would be the ratio of the posterior and prior normalization factors.

8. What is a hyper-hyper-parameter?

Answer: The prior put on the hyper-parameters is called hyper-hyper-parameter or hyper-prior.

9. Suppose we use the tabular parameterization for multi-class classification with d features, where we have k classes and each feature can take k values. What is the exact number of parameters we need if we remove redundant parameters by exploiting that probabilities sum to 1.

Answer: Using the fact that probabilities sum to 1, for fixed features, we need $k - 1$ parameters. Therefore, in total, we have $(k - 1) * k^d$ parameters.

10. In the softmax function we have k weight vectors, one for each class. But in the special case of the sigmoid function we fix one of these weight vectors to zero, so we only have one weight vector for two classes. Explain why we would or would not get a more-general model if we used the softmax function with two weight vectors for binary classification (by “more-general”, we mean “can model a larger set of functions”).

Answer: We do not get a “more-general” model by using two weight vectors with the softmax function for binary classification. If we do, we get $f_1(z_1, z_2) = \frac{\exp(z_1)}{\exp(z_1) + \exp(z_2)}$ and $f_2(z_1, z_2) = \frac{\exp(z_2)}{\exp(z_1) + \exp(z_2)}$. We note that $f_1(z_1, z_2) = 1 - f_2(z_1, z_2)$ and vice versa. Therefore, using both weight vectors is redundant.

11. If we use a neural network with one hidden layer for multi-class classification, what are the sizes of the parameter matrices W and V and what is the cost of backpropagation? You can use k as the number of classes and m as the number of hidden units.

Answer: Assuming that our samples are d -dimensional: W is $m \times d$, and V is $k \times m$ dimensional (that is md and km parameters, respectively). Using automatic differentiation, the cost of gradient computation is the same as forward propagation. Therefore, the cost is $O(md + km)$

12. Why can RNNs label sequence of different lengths?

Answer: This is possible since RNNs use the same parameters accross time, that is, they use parameter tying, and therefore, RNNs can handle variable sequence lengths.

13. When predicting with sequence-to-sequence RNNs we do not know the length of the output sequence. How is the length of the output sequence determined during decoding?

Answer: Through the use of a special tokens indicating beginning and ending of sequence denoted by "BOS" and "EOS" respectively, we know when to start and stop the decoding process.

14. In LSTMs, what is the range (set of possible values) for a_t , o_t , c_t , f_t , i_t , and g_t under the usual choices of activation functions (h : sigmoid, h_0 : tanh)?

Answer:

- a_t : $(-1, 1)$
- o_t : $(0, 1)$
- c_t : $(-\infty, +\infty)$
- f_t : $(0, 1)$
- i_t : $(0, 1)$
- g_t : $(-1, 1)$

15. Why do we use "context vectors" instead of including all encoding states in attention models?

Answer: Because weighting all encoder states would mean that we need the number of inputs to be the same for all samples. In other words, the number of decoding weights depend on input size, and therefore, this size would have to be fixed, which is impractical.

Project Proposal(s)

As discussed in the syllabus, we are using the following marking scheme for the course:

1. CPSC 440: 50% assignments and 50% for the best among {final exam, research project, sample lecture/assignment}.
2. CPSC 540: 50% assignments, 25% for the best among {final exam, research project, sample lecture/assignment}, and 25% for the second best among those three categories.

If you are doing the research project and/or the sample lecture/assignment, [the final part of this assignment is to submit a short proposal about what you will focus on](#). Both the project and sample lecture/assignment must be done in groups of 2-3, but you should only submit one proposal for the group. The next two subsections describe what the proposal should look like for the two types of projects. This part of the assignment will not be marked: we are doing this so that you (a) start forming project groups now, (b) you start thinking about what you want to do for the project(s), and (c) we check that the scope and topic of the project is suitable.

If you plan to do the final exam but not the research project or sample lecture/assignment, then you do not need to submit anything for this part of the assignment. If you are doing the research project and sample lecture/assignment, note that the groups do not have to be the same for both. Also note that you can do these projects on similar topics. For example, the research project might focus on applying a particular ML method to a particular application while the lecture goes over how the ML method works. Similarly, if you have projects in other courses this term, it is ok to do them on the same topic provided that you check with the other instructor (although the submission formats and grading schemes may differ between courses).

There is flexibility in the choice of project topics even after the proposal. If you want to explore different topics you can ultimately choose to do a project that is unrelated to the one in your proposal. However, in this case you may want to check with the instructor/TAs that the scope and topic are appropriate..

Research Project

The proposal should be a [maximum of 2 pages](#) (and 1 page or half of a page is ok if you can describe your plan concisely). The proposal should be written for the instructor and the TAs, so you do not need to introduce any ML background but you will need to introduce non-ML topics. The projects [must be done in groups of 2-3](#), and the [proposal be submitted separately from the assignment](#) on Gradescope.

There is quite a bit of flexibility in terms of the type of project you can do, as I believe there are many ways that people can make valuable contributions to research. However, note that the final deliverable will be a report containing at most 6 pages of text (the actual document can be longer due to figures, tables, references, and proofs) that emphasizes a particular “contribution” (which is “what doing the project has added to the world”).

[The three main questions your project proposal needs to answer are:](#)

1. [What problem you are focusing on?](#)
2. [What do you plan to do?](#)
3. [What will be the “contribution”?](#)

Also, note that for the course project that negative results (like “we tried something that we thought we would work in a particular setting but it didn’t work”) are acceptable (and often unavoidable).

Here are some standard project “templates” that you might want to follow:

- **Application bake-off:** you pick a specific application (from your research, personal interests, or maybe from Kaggle) or a small number of related applications, and try out a bunch of techniques (e.g., random forests vs. logistic regression vs. generative models). In this case, the contribution would be showing that some methods work better than others for this specific application (or your contribution could be that everything works equally well/badly).
- **New application:** you pick an application where people aren't using ML, and you test out whether ML methods are effective for the task. In this case, the contribution would be knowing whether ML is suitable for the task (and perhaps how to prepare the data and construct features).
- **Scaling up:** you pick a specific machine learning technique, and you try to figure out how to make it run faster or on larger datasets (for example, how do we apply kernel methods when n is very large). In this case, the contribution would be the new technique and an evaluation of its performance, or could be a comparison of different ways to address the problem.
- **Improving performance:** you pick a specific machine learning technique, and try to extend it in some way to improve its performance (for example, how can we efficiently use non-linearities within graphical models). In this case, the contribution would be the new technique and an evaluation of its performance.
- **Generalization to new setting:** you pick a specific machine learning technique, and try to extend it to a new setting (for example, making a graphical-model version of random forests). In this case, the contribution would be the new technique and an evaluation of its performance, or could be a comparison of different ways to address the problem.
- **Perspective paper:** you pick a specific topic in ML, read a larger number of papers on the topic, then write a report summarizing what has been done on the topic and what are the most promising directions of future work. In this case, the contribution would be your summary of the relationships between the existing works, and your insights about where the field is going.
- **Coding project:** you pick a specific method or set of methods (like independent component analysis), and build an implementation of them. In this case, the contribution could be the implementation itself or a comparison of different ways to solve the problem.
- **Theory:** you pick a theoretical topic (like the variance of cross-validation or the convergence of stochastic gradient in non-smooth and non-convex setting), read what has been done about it, and try to prove a new result (usually by relaxing existing assumptions or adding new assumptions). The contribution could be a new analysis of an existing method, or why some approaches to analyzing the method will not work.

The above are just suggestions, and many projects will mix several of these templates together, but if you are having trouble getting going then it's best to stick with one of the above templates. Also note that the above includes topics not covered in the course (like random forests), so there is flexibility in the topic, but the topic should be closely-related to ML.

Sample Lecture and Assignment

Throughout CPSC 340-440, we try to introduce you to a variety of fundamental concepts that we think will stand the test of time and we also try to give you an idea of what methods people are currently finding useful in a variety of settings. But unfortunately, ML is a huge field so many topics cannot be covered in only 1-2 courses. For the sample lecture/assignment, you will [prepare a lecture and an assignment on a model that we do not cover in the course](#) (preparing material is one of the best ways to learn new things, and this also helps us decide what topics to include in future offerings of the course).

The lecture should touch on most of the below topics (I planned each "part" of the course this term to follow this structure):

1. **Motivating problem:** introduce an application that motivates the model.
2. **Model definition:** how is it defined and what are the parameters (and their sizes)?
3. **General framework and other applications:** is this solving an abstract problem, and where else would this model be useful?
4. **Inference:** how do you do things like sampling, decoding, marginalization, conditioning, and test-set predictions/evaluations? (Theoretically and with code.)
5. **MLE:** how do you compute the MLE parameters? (Theoretically and with code.)
6. **MAP:** how do you introduce a prior and compute the MAP parameters?
7. **Bayes:** how do you make Bayesian predictions with the model?
8. **Multivariate** (for one-dimensional distributions): is there a multi-variable version of the model?
9. **MNIST:** what do MNIST samples look like if you use it as a density estimator?
10. **Supervised:** how do we add features to the model and use within a generative or a discriminative classifier?
11. **Deep:** how do we add layers of hidden layers to learn features?

The particular deliverables due with this assignment are:

1. [Specify which model you will focus on.](#)
2. [Give a 1-sentence-maximum description of how you might cover the topics above in the model.](#) Note: not all topics make sense for all models, so it's ok to say that you won't cover some of them.
3. [Give a short outline for what the assignment related to the model will cover.](#)

As with the research project, [the sample lecture/assignment should be done in groups of 2-3](#) and the [deliverables above should be submitted separately from the assignment](#). The appendix below may help you to choose a topic.

A Lecture Topic Suggestions

Below are a set of topics that I plan to cover later in the course, so you should avoid choosing these topics:

- Multivariate Gaussians and models like linear/Gaussian discriminant analysis.
- Exponential families and multivariate Laplace/student distributions.
- Rejection sampling and importance sampling.
- Markov chains and MCMC.
- Directed and undirected graphical models (“Bayesian networks” and “Markov random fields”), conditional random fields.
- Latent Dirichlet allocation and variational inference.
- Mixture models and hidden Markov models.
- Boltzmann machines and deep belief networks.

I may also cover VAEs/GANs if we have time, but feel to choose those topics as I am not sure if we will have time and these topics are changing very quickly at the moment.

Below is a set of topics that you might see in other ML courses that I will probably not cover as well as some related keywords:

- Sequential Monte Carlo.
- Non-parameteric Bayes.
- Online learning and bandits.
- Reinforcement learning.
- Privacy and security.
- Algorithm fairness.
- Independent component analysis.
- Scaling up Gaussian processes.
- Max-margin Markov networks and structured SVMs.
- Reinforcement learning.
- Causality.
- Parallel/distributed/federated training.
- Learning theory.
- Probabilistic context-free grammars.
- Probabilistic programming.
- Sub-modularity.
- Spectral methods.
- Automatic hyper-parameter tuning.

(The above topics would probably be my starting point for making a third class, and I could imagine some of the above topics being swapped into 340/440 in future offerings.) Some of these topics are too big for one lecture (such as reinforcement learning), and one of the main purposes of doing this proposal is to help you choose the appropriate scope.

CPSC 440/540 Machine Learning (January-April, 2022)

Assignment 4 (due Friday April 1st at midnight)

For this assignment you can work in groups of 1-2. However, please only hand in one assignment for the group. It is possible that some questions on this assignment will be cancelled, depending on the pace of lectures.

1. Name(s):

Answer: Masoud Mokhtari, Ali Mehrabian

2. Student ID(s):

Answer: 14186167, 31662323

1 Gaussians

1.1 Gaussian Self-Conjugacy

Consider n IID samples x^i distributed according to a Gaussian with mean μ and covariance $\sigma^2 I$,

$$x^i \sim \mathcal{N}(\mu, \sigma^2 I).$$

Assume that μ itself is distributed according to a Gaussian

$$\mu \sim \mathcal{N}(\mu_0, \Sigma_0),$$

with mean μ_0 and (positive-definite) covariance Σ_0 . In this setting, the posterior for μ also follows a Gaussian distribution.

Derive the form of the posterior distribution, $p(\mu \mid X, \sigma^2, \mu_0, \Sigma_0)$.

Hint: the posterior is a product of Gaussian densities.

Answer:

$$p(\mu \mid X, \sigma^2, \mu_0, \Sigma_0) \propto p(\mu \mid \mu_0, \Sigma_0) \prod_{i=1}^n p(x^i \mid \mu, \sigma^2 I)$$

Now, since in the Gaussian form, there is a squared term like $(x^i - \mu)^2$, we can switch μ and x^i without changing the form:

$$p(\mu \mid X, \sigma^2, \mu_0, \Sigma_0) \propto p(\mu \mid \mu_0, \Sigma_0) \prod_{i=1}^n p(\mu \mid x^i, \sigma^2 I)$$

To solve this, we can use *Product of Gaussian Densities Property*, which states that for $p(x) \propto f_1(x)f_2(x)$ with f_1 and f_2 being Gaussians with μ_1, μ_2 and Σ_1, Σ_2 respectively, p is a Gaussian with $\Sigma = (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1}$ and $\mu = \Sigma \Sigma_1^{-1} \mu_1 + \Sigma \Sigma_2^{-1} \mu_2$. Therefore, we have:

$$\mu \mid X, \sigma^2, \mu_0, \Sigma_0 \sim \mathcal{N}(\mu^+, \Sigma^+)$$

Where:

- $\Sigma^+ = (\Sigma_0^{-1} + \frac{n}{\sigma^2} I)^{-1}$
- $\mu^+ = \Sigma^+ (\Sigma_0^{-1} \mu_0 + \sum_{i=1}^n \frac{1}{\sigma^2} I x^i) = \Sigma^+ (\Sigma_0^{-1} \mu_0 + \frac{1}{\sigma^2} I \sum_{i=1}^n x^i) = \Sigma^+ (\Sigma_0^{-1} \mu_0 + n \frac{\mu_{MLE}}{\sigma^2} I)$

1.2 Generative Classifiers with Gaussian Assumption

Consider the 3-class classification dataset in this image: In this dataset, we have 2 features and each colour represents one of the classes. Note that the classes are highly-structured: the colours each roughly follow a Gaussian distribution plus some noisy samples.

Since we have an idea of what the features look like for each class, we might consider classifying inputs x using a *generative classifier*. In particular, we are going to use Bayes rule to write

$$p(y^i = c \mid x^i, \Theta) = \frac{p(x^i \mid y^i = c, \Theta) \cdot p(y^i = c \mid \Theta)}{p(x^i \mid \Theta)},$$

where Θ represents the parameters of our model. To classify a new example \tilde{x}^i , generative classifiers would use

$$\hat{y}^i = \arg \max_{y \in \{1, 2, \dots, k\}} p(\tilde{x}^i \mid y^i = c, \Theta) p(y^i = c \mid \Theta),$$

where in our case the total number of classes k is 3.¹ Modeling $p(y^i = c \mid \Theta)$ is easy: we can just use a k -state categorical distribution,

$$p(y^i = c \mid \Theta) = \theta_c,$$

where θ_c is a single parameter for class c . The maximum likelihood estimate of θ_c is given by n_c/n , the number of times we have $y^i = c$ (which we've called n_c) divided by the total number of data points n .

Modeling $p(x^i \mid y^i = c, \Theta)$ is the hard part: we need to know the *probability of seeing the feature vector x^i given that we are in class c* . This corresponds to solving a density estimation problem for each of the k possible classes. To make the density estimation problem tractable, we'll assume that the distribution of x^i given that $y^i = c$ is given by a $\mathcal{N}(\mu_c, \Sigma_c)$ Gaussian distribution for a class-specific μ_c and Σ_c ,

$$p(x^i \mid y^i = c, \Theta) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma_c|^{\frac{1}{2}}} \exp \left(-\frac{1}{2} (x^i - \mu_c)^T \Sigma_c^{-1} (x^i - \mu_c) \right).$$

Since we are distinguishing between the probability under k different Gaussians to make our classification, this is called *Gaussian discriminant analysis* (GDA). In the special case where we have a constant $\Sigma_c = \Sigma$ across all classes it is known as *linear discriminant analysis* (LDA) since it leads to a linear classifier between any two classes (while the region of space assigned to each class forms a convex polyhedron as in k -means clustering and softmax classification). Another common restriction on the Σ_c is that they are diagonal matrices, since this only requires $O(d)$ parameters instead of $O(d^2)$ (corresponding to assuming that the features are independent univariate Gaussians given the class label). Given a dataset $\mathcal{D} = \{(x^i, y^i)\}_{i=1}^n$, where $x^i \in \mathbb{R}^d$ and $y^i \in \{1, \dots, k\}$, the maximum likelihood estimate (MLE) for the μ_c and Σ_c in the GDA model is the solution to

$$\arg \max_{\mu_1, \mu_2, \dots, \mu_k, \Sigma_1, \Sigma_2, \dots, \Sigma_k} \prod_{i=1}^n p(x^i \mid y^i, \mu_{y^i}, \Sigma_{y^i}).$$

This means that the negative log-likelihood will be equal to

$$\begin{aligned} -\log p(X \mid y, \Theta) &= -\sum_{i=1}^n \log p(x^i \mid y^i, \mu_{y^i}, \Sigma_{y^i}) \\ &= \sum_{i=1}^n \frac{1}{2} (x^i - \mu_{y^i})^T \Sigma_{y^i}^{-1} (x^i - \mu_{y^i}) + \frac{1}{2} \sum_{i=1}^n \log |\Sigma_{y^i}| + \text{const.} \end{aligned}$$

In class we stated the MLE for this model under the assumption that we use full covariance matrices and that each class has its own covariance.

¹The denominator $p(\tilde{x}^i \mid \Theta)$ is irrelevant to the classification since it is the same for all y .

1. Derive the MLE for the GDA model under the assumption of *common diagonal covariance* matrices, $\Sigma_c = D$ (d parameters). (Each class will have its own mean μ_c .)

Answer: Let's find the MLE for μ_c first:

$$\nabla_{\mu_c} - \log p(X | y, \Theta) = \nabla_{\mu_c} - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^d \frac{1}{\sigma_j^2} (x_j^i - \mu_{jy^i})^2 = \sum_{y^i=c}^n \sum_{j=1}^d \frac{1}{\sigma_j^2} (x_j^i - \mu_{jy^i}) = \sum_{j=1}^d \frac{1}{\sigma_j^2} \sum_{y^i=c}^n (x_j^i - \mu_{jc})$$

Setting the above gradient to 0, we have $-n_c \mu_c + \sum_{y^i=c} x^i = 0$. **Therefore, the MLE is $\hat{\mu}_c = \frac{1}{n_c} \sum_{y^i=c} x^i$.**

Let's find the MLE for Σ_c . Since we have a common Σ_c between classes, the y^i subscript is dropped:

$$\nabla_{\Sigma_c} - \log p(X | y, \Theta) = \nabla_{\Sigma_c} \frac{1}{2} \sum_{i=1}^n (x^i - \mu_{y^i})^T \Sigma^{-1} (x^i - \mu_{y^i}) + \frac{1}{2} \sum_{i=1}^n \log |\Sigma|$$

We use the re-parametrization trick shown in the tutorial, where we replace Σ^{-1} with Θ :

$$= \nabla_{\Sigma_c} \frac{1}{2} \sum_{i=1}^n (x^i - \mu_{y^i})^T \Theta (x^i - \mu_{y^i}) + \frac{1}{2} \sum_{i=1}^n \log |\Theta^{-1}|$$

Since Θ is a diagonal matrix and since all samples use the same Θ , we have:

$$= \nabla_{\Sigma_c} \frac{1}{2} \sum_{i=1}^n (x^i - \mu_{y^i})^T \Theta (x^i - \mu_{y^i}) - \frac{n}{2} \log |\Theta|$$

As shown in the lecture, for a scalar $y^T A y$, we have $y^T A y = \text{Tr}(y^T A y)$. And using the cyclic property of the trace operator, we have $y^T A y = \text{Tr}(y^T A y) = \text{Tr}(y y^T A)$ conditioned on dimensions matching (which is the case above):

$$= \nabla_{\Sigma_c} \frac{1}{2} \sum_{i=1}^n \text{Tr}((x^i - \mu_{y^i})^T (x^i - \mu_{y^i}) \Theta) - \frac{n}{2} \log |\Theta|$$

Since both trace and summation are linear operations, we can exchange them:

$$= \nabla_{\Sigma_c} \frac{1}{2} \text{Tr}(\Theta \sum_{i=1}^n (x^i - \mu_{y^i})^T (x^i - \mu_{y^i})) - \frac{n}{2} \log |\Theta|$$

Using $\nabla_{\Theta} \text{Tr}(A \Theta) = A$ and $\nabla_{\Theta} \log |\Theta| = \Theta^{-1}$:

$$= \frac{1}{2} \sum_{i=1}^n (x^i - \mu_{y^i})^T (x^i - \mu_{y^i}) - \frac{n}{2} \Theta^{-1}$$

Setting the above to 0 and finding Θ , we get $\hat{\Theta}^{-1} = \hat{\Sigma}_c = \frac{1}{n} \sum_{i=1}^n (x^i - \mu_{y^i})^T (x^i - \mu_{y^i})$.

2. Derive the MLE for the GDA model under the assumption of *individual scaled-identity* matrices, $\Sigma_c = \sigma_c^2 I$ (k parameters).

Answer: For μ_c , similar to the previous part, we have:

$$\nabla_{\mu_c} - \log p(X | y, \Theta) = \nabla_{\mu_c} - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^d \frac{1}{\sigma_c^2} (x_j^i - \mu_{jy^i})^2 = \sum_{y^i=c}^n \frac{1}{\sigma_c^2} (x_j^i - \mu_{jy^i}) = \frac{1}{\sigma_c^2} \sum_{y^i=c}^n (x^i - \mu_c)$$

Therefore, $\hat{\mu}_c = \frac{1}{n_c} \sum_{y^i=c}^n x^i$.

For the MLE of Σ_c , using the same re-parametrization and trace trick and using the fact that our covariance matrix is diagonal, we have:

$$\begin{aligned}
&= \nabla_{\Sigma_c} \frac{1}{2} \sum_{i=1}^n (x^i - \mu_{y^i})^T \Theta_{y^i} (x^i - \mu_{y^i}) + \frac{1}{2} \sum_{i=1}^n \log |\Theta_{y^i}^{-1}| \\
&= \nabla_{\Sigma_c} \frac{1}{2} \text{Tr}(\Theta_{y^i} \sum_{i=1}^n (x^i - \mu_{y^i})^T (x^i - \mu_{y^i})) - \frac{1}{2} \sum_{i=1}^n \log |\Theta_{y^i}| \\
&= \frac{1}{2} \sum_{y^i=c} (x^i - \mu_{y^i})^T (x^i - \mu_{y^i}) - \frac{n_c}{2} \Theta^{-1}
\end{aligned}$$

Setting the above to 0 and finding Θ , we get $\hat{\Theta}^{-1} = \hat{\Sigma}_c = \frac{1}{n_c} \sum_{y^i=c} (x^i - \mu_{y^i})^T (x^i - \mu_{y^i})$,

3. When you run *example_generative* it loads a variant of the dataset in the figure that has 12 features and 10 classes. This data has been split up into a training and test set, and the code fits a k -nearest neighbour classifier to the training set then reports the accuracy on the test data (around $\sim 63\%$ test error). The k -nearest neighbour model does poorly here since it doesn't take into account the Gaussian-like structure in feature space for each class label. Write a function *gda* that fits a GDA model to this dataset (using individual full covariance matrices). [Hand in the function and report the test set accuracy.](#)

Answer: The code shown in Figure 1 was used to achieve a test error of 36.9% and test accuracy of 63.1%.

```
function gda(X,y,k)
# Implementation of GDA

# Center the data first
Xmean = mean(X, dims=1)
X = X .- Xmean

# Extract number of samples and their dimension
d = size(X, 2)
n = size(X, 1)

# Initialize parameters
sigmas = zeros((k, d, d))
mus = zeros((k, d))
pis = zeros((k, 1))

# Go through data and extract required parameters
# for each class
for c in 1:k
    nc = sum(y .== c)
    pis[c] = nc / n
    mus[c, :] = sum(X[y.==c, :], dims=1) ./ nc
    sigmas[c, :, :] = cov(X[y.==c, :])
end

function predict(Xhat)
# Center the data
Xhat = Xhat .- Xmean

t = size(Xhat, 1)
yhat = zeros(t)

# For each sample, find its probability under the class parameters
for i in 1:t
    max_prob = -9999999
    max_prob_class = 0
    for c in 1:k
        d = MvNormal(mus[c, :], sigmas[c, :, :])
        prob = log(pis[c]) + logpdf(d, Xhat[i, :])
        if prob > max_prob
            max_prob = prob
            max_prob_class = c
        end
    end
    yhat[i] = max_prob_class
end

return yhat
end

return GenericModel(predict)
end
```

Figure 1: GDA implementation in Julia

4. In this question we would like to replace the Gaussian distribution of the previous problem with the more robust multivariate-t distribution so that it isn't influenced as much by the noisy data. Unlike the previous case, we don't have a closed-form solution for the parameters. However, if you run *example_student* it generates random noisy data and fits a multivariate-t model. By using the *studentT* model, write a new function *tda* that implements a generative model that is based on the multivariate-t distribution instead of the Gaussian distribution. [Report the test accuracy with this model.](#)

Answer: The code shown in Figure 2 was used to achieve a test error of 19.6% and test accuracy of 80.4%.

```
function tda(X,y,k)
# Implementation of TDA

# Center the data first
Xmean = mean(X, dims=1)
X = X .- Xmean

# Extract number of samples and their dimension
d = size(X, 2)
n = size(X, 1)

# Initialize array containing the studentT dist for each class
subModel = Array{DensityModel}(undef,k)
pis = zeros((k, 1))

# Go through data and create the dist object for each class
for c in 1:k
    pis[c] = sum(y .== c) / n
    subModel[c] = studentT(X[y.==c, :])
end

function predict(Xhat)
# Center the data
Xhat = Xhat .- Xmean

t = size(Xhat, 1)
probs = zeros((t, k))

# For each sample, find its probability under the class parameters
for c in 1:k
    probs[:, c] = log(pis[c]) .+ log.(subModel[c].pdf(Xhat))
end
yhat = mapslices(argmax, probs, dims=2)

return yhat
end

return GenericModel(predict)
end
```

Figure 2: TDA implementation in Julia

Hints: you may be able to substantially simplify the notation in the MLE derivations if you use the notation $\sum_{i \in y_c}$ to mean the sum over all values i where $y^i = c$. Similarly, you can use n_c to denote the number of cases where $y_i = c$, so that we have $\sum_{i \in y_c} 1 = n_c$. Note that the determinant of a diagonal matrix is the product of the diagonal entries, and the inverse of a diagonal matrix is a diagonal matrix with the reciprocals of the original matrix along the diagonal.

For the implementation you can use the result from class regarding the MLE of a general multivariate Gaussian. At test time for GDA, you may find it more numerically reasonable to compare log probabilities rather than probabilities of different classes, and you may find it helpful to use the *logdet* function to compute the log-determinant in a more numerically-stable way than taking the log of the determinant. (Also, don't forget to center at training and test time.)

For the last question, you may find it helpful to define an empty array that can be filled with k *DensityModel* objects using:

```
subModel = Array{DensityModel}(undef,k)
```

1.3 Empirical Bayes

Consider the model

$$y^i \sim \mathcal{N}(w^T z^i, \sigma^2), \quad w_j \sim \mathcal{N}(0, \lambda^{-1}),$$

where z^i is a length- k non-linear transformation of the features x^i (like a polynomial basis or RBFs). The posterior distribution in this model has the form

$$w \sim \mathcal{N}(w^+, \Theta^{-1}),$$

where the posterior precision and mean are given by

$$\Theta = \frac{1}{\sigma^2} Z^T Z + \lambda I,$$

$$w^+ = \frac{1}{\sigma^2} \Theta^{-1} Z^T y,$$

and Z contains the z^i vectors in the rows. The marginal likelihood in this model is given by

$$p(y | X, \sigma^2, \lambda) = \frac{(\lambda)^{k/2}}{(\sigma\sqrt{2\pi})^n |\Theta|^{1/2}} \exp \left(-\frac{1}{2\sigma^2} \|Zw^+ - y\|^2 - \frac{\lambda}{2} \|w^+\|^2 \right).$$

As discussed in class, the marginal likelihood can be used to optimize hyper-parameters like σ , λ , and even the basis Z .

The demo *example_basis* loads a dataset and fits a degree-2 polynomial to it. Normally we would use a test set to choose the degree of the polynomial but here we'll use the marginal likelihood of the training set. Write a function, *leastSquaresEmpiricalBaysis*, that uses the marginal likelihood to choose the degree of the polynomial as well as the parameters λ and σ (you can restrict your search for λ and σ to powers of 2). **Hand in your code and report the marginally most likely values of the degree, σ , and λ .**

Answer: The code is shown in Figure 3, and most likely values are $\sigma = 16$, $\lambda = 2$ and degree = 4.

```
function leastSquaresEmpiricalBaysis(x, y, max_p, max_sigma, max_lambda)

    marg_likes = zeros((max_p, floor(Int, log2(max_lambda) + 1), floor(Int, log2(max_sigma) + 1)))

    count = 1
    for p in 1:max_p
        for sig in 0:floor(Int, log2(max_sigma))
            for lam in 0:floor(Int, log2(max_lambda))

                # Using powers of 2 for lambda and sigma
                sigma = 2^sig
                lambda = 2^lam

                # Create the polynomial basis
                Z = polyBasis(x, p)

                n, k = size(Z)

                # Find the log marginal likelihood
                Theta = 1/sigma^2 * Z' * Z + lambda * I
                wp = 1/sigma^2 * inv(Theta) * Z' * y
                marg_likes[p, lam+1, sig+1] = k/2*log(lambda) - n * log(sigma) - n * log(sqrt(2*n)) - 1/2 *
                logdet(Theta) + (-1/(2*sigma^2) * norm(Z * wp - y, 2)^2 - lambda/2 * norm(wp, 2)^2)

                count += 1
            end
        end
    end

    max = maximum(marg_likes)
    maxidx = findall(x->x==max, marg_likes)[1]
    sigma = 2^maxidx[3]
    lambda = 2^maxidx[2]
    p = maxidx[1]

    @printf("Chosen P = %f, Chosen Sigma = %f, Chosen Lambda = %f \n", p, sigma, lambda)
```

Figure 3: Empirical Bayes Implementation for Gaussian Regression

2 Markov Models

2.1 Inference with Discrete States

The function *example_markov.jl* loads the initial state probabilities and transition probabilities for a Markov chain model,

$$p(x_1, x_2, \dots, x_d) = p(x_1) \prod_{j=2}^d p(x_j | x_{j-1}),$$

corresponding to the “grad student Markov chain” from class.

1. Write a function, *sampleAncestral*, that uses ancestral sampling to sample a sequence x from this Markov chain of length d . [Hand in this code and report the univariate marginal probabilities for time 50 using a Monte Carlo estimate based on 10000 samples.](#)

Hint: you can use *sampleDiscrete* in *misc.jl* to sample from a discrete probability mass function using the inverse transform method.

Answer: The code is shown in Figures 4 and 5, and the marginal probabilities at time 50 are summarized below:

- k=1: 0.0155
- k=2: 0.3816
- k=3: 0.0156
- k=4: 0.0069
- k=5: 0.1023
- k=6: 0.0890
- k=7: 0.3891

```
80 function sampleAncestral(p1, pt, s, d)
81     # Performs s ancestral samplings of a markov chain for d transitions
82     samples = zeros((s, d))
83
84     for simulation in 1:s
85         # Initial state sampling
86         samples[simulation, 1] = sampleDiscrete(p1)
87
88         # Generate samples ancestrally using the transition probabilities
89         for step in 2:d
90             samples[simulation, step] = sampleDiscrete(pt[floor(Int, samples[simulation, step-1]), :])
91         end
92     end
93
94     return samples
95 end
```

Figure 4: Julia implementation for the sampleAncestral function

```
18 last_step = 50
19 num_sims = 10000
20
21 # Generate 10000 samples of length 50
22 samples = sampleAncestral(p1, pt, num_sims, last_step)
23
24 # Extract the last column
25 state_of_interest = samples[:, last_step]
26
27 # Find the marginal probabilities
28 for i in 1:k
29     n_i = sum(state_of_interest .== i)
30     @printf("Marginal probability of %d th category is %f \n", i, n_i/num_sims)
31 end
```

Figure 5: Using ancestral sampling and MC estimation for markov chains

2. Write a function, *marginalCK*, that uses the CK equations to compute the exact univariate marginals up to a given time d . Hand in this code, report all exact univariate marginals at time 50, and report how this differs from the marginals in the previous question.

Answer: The code is shown in Figures 6 and 7 and the marginal probabilities are summarized below. We see that Monte Carlo estimates are very close to the exact marginals found using the CK equations. This could be due to the simplistic nature of our setting where MC estimates are converging at a fast rate.

- k=1: 0.0139
- k=2: 0.3863
- k=3: 0.0170
- k=4: 0.0054
- k=5: 0.1041
- k=6: 0.0843
- k=7: 0.3889

```

22 function marginalCK(p1, pt, d)
23     # Finds exact marginal at step d
24     marginals = pt' * p1
25
26     for step in 1:d-2
27         marginals = pt' * marginals
28     end
29
30     return marginals
31 end

```

Figure 6: Julia implementation for the marginalCK function

```

33 last_step = 50
34
35 # Use CK to find exact marginals
36 marginals = marginalCK(p1, pt, last_step)
37
38 # Print the marginal probabilities
39 for i in 1:k
40     @printf("CK Marginal probability of %d th category is %f \n", i, marginals[i])
41 end

```

Figure 7: Finding marginals at step 50 using CK equations

3. What is the state c with highest marginal probability, $p(x_j = c)$, for each time j ?

Answer: Using the code shown in Figure 8, we see that for steps 1 to 49, the second category or "industry" has the highest marginal probability, while from step 50 onwards, category 7 or "death" (which is also the stationary state of the chain) has the highest probability.

```

43 # Find c with highest probability at each step
44 for last_step in 1:100
45     marginals = marginalCK(p1, pt, last_step)
46     @printf("c with max marginal probability at step %d is %d \n", last_step, argmax(marginals))
47 end

```

Figure 8: Highest marginal probability at each step.

[illegible]

Figure 11: Decoded chain for $d=100$.

5. Report all the univariate conditional probabilities at time 50 if the student starts in grad school, $p(x_{50} = c \mid x_1 = 3)$ for all c . Hint: you should be able to do this by changing the input to the CK equations.

Answer: Adding conditioning to CK amounts to enforcing an initial state by giving it a probability of 1. Therefore, we change `p1` to `[0, 0, 1, 0, 0, 0, 0]` and use our code from part 2 to get the following conditional probabilities at time 50:

- `c=1:` 0.00834
- `c=2:` 0.23112
- `c=3:` 0.01019
- `c=4:` 0.00849
- `c=5:` 0.18312
- `c=6:` 0.16988
- `c=7:` 0.38888

6. Report for all c the univariate conditional probabilities $p(x_5 = c \mid x_{10} = 6)$ (“where you were likely to be 5 years after graduation if you ended up in academia after 10 years”) obtained using a Monte Carlo estimate based on 10000 samples and rejection sampling. Also report the number of samples accepted among the 10000 samples.

Answer: Accepting only 412 out of 10000 samples, the following probabilities were found for 5th state:

- `c=1:` 0.00254
- `c=2:` 0.03817
- `c=3:` 0.24173
- `c=4:` 0.06107
- `c=5:` 0.16285
- `c=6:` 0.49364
- `c=7:` 0.00000

7. Give code implementing a dynamic programming approach to exactly compute $p(x_5 = c \mid x_{10} = 6)$, and report the exact values for all c .

Answer: The code is shown in Figure 12 and the probabilities are summarized below:

- `c=1:` 0.00155
- `c=2:` 0.03393
- `c=3:` 0.24774
- `c=4:` 0.05807
- `c=5:` 0.16134
- `c=6:` 0.49737
- `c=7:` 0.00000

```

function fb_algorithm(p1, pt, k, d, cond, j)

    # initialize the M matrix
    M = zeros((k, d))
    M[:, 1] = p1

    # initialize phi
    phi = ones((k, d))
    phi[:, 1] = p1

    # Conditioning
    phi[:, d] = zeros(k)
    phi[cond, d] = 1

    # Initialize the V matrix
    V = zeros((k, d))
    V[:, d] = phi[:, d]

    # Populate the M matrix
    for step in 2:d
        for xj in 1:k
            for xjprev in 1:k
                M[xj, step] = M[xj, step] + phi[xj, j] * pt[xjprev, xj] * M[xjprev, step-1]
            end
        end
    end

    # Normalizing constant
    Z =

    # Populate the V matrix
    for step in d-1:-1:1
        for xj in 1:k
            for xjnext in 1:k
                V[xj, step] = V[xj, step] + phi[xj, j] * pt[xj, xjnext] * V[xjnext, step+1]
            end
        end
    end

    prob = M[:, j] .* V[:, j] ./ sum(M[:, j] .* V[:, j])
    return prob
end

```

Figure 12: Julia implementation of forward backward algorithm for future conditioning.

8. Why is $p(x_j = 7 \mid x_{10} = 6)$ equal to zero for all j less than 10?

Answer: The 7th category corresponds to "death". If you are in academia 10 years after graduation, you cannot be dead before that unless ghosts are haunting ICICS and teaching courses. More concretely, once you enter the "dead" state, you stay in that state, so there is no way to enter academia once you are "dead", that is $p(x_j = 10 \mid x_{j-1} = 7) = 0$ and $p(x_j = 7 \mid x_{j-1} = 7) = 1$.

Hint: for some of the questions you may find it helpful to use a k by d matrix M to represent a dynamic programming table

2.2 Markov Chain Monte Carlo

If you run *example.MH.jl*, it loads a set of images of ‘2’ and ‘3’ digits. It then runs the Metropolis MCMC algorithm to try to generate samples from the posterior over w , in a logistic regression model with a Gaussian prior. Once the samples are generated, it makes a histogram of the samples for several of the variables.²

1. Why would the samples coming from the Metropolis algorithm not give a good approximation to the posterior?

Answer: In this case, we see that none of the samples are being accepted, and we simply keep the prior distribution. This is due to a standard deviation of 1 being too high for the noises that are added to W throwing it far off from the logistic objective, causing all samples to be rejected.

A more general answer regarding Metropolis algorithm: When using samples from our Metropolis algorithm, we are not certain if we have reached the desired stationary distribution. To slightly improve this situation, we can use “Burn In” or “Thinning” where we ignore the initial samples or take every k_{th} sample, respectively. However, in this case, we are not using any such strategies. Additionally, our proposed samples are generated by addition of an unsuitable Gaussian noise, causing most samples to be rejected and not converging to the posterior.

2. Modify the proposal used by the demo to $\hat{w} \sim \mathcal{N}(w, (1/100)I)$ instead of $\hat{w} \sim \mathcal{N}(w, I)$. [Hand in your code and the update histogram plot.](#)

Answer: The code and histogram are shown in Figures 13 and 14, respectively.

```
3 function blogreg(X,y,lambda,nSamples)
4
5     (n,d) = size(X);
6
7     samples = zeros(nSamples,d)
8
9     # Initialize and compute negative log-posterior (up to constant)
10    w = zeros(d,1);
11    log_p = logisticObj(w,X,y,lambda)
12
13    nAccept = 0
14    for s in 1:nSamples
15
16        # Propose candidate
17        noise = 1/100 * randn(d,1)
18        wHat = w + noise
19        log_phat = logisticObj(wHat,X,y,lambda)
20
21        # Metropolis-Hastings accept/reject step (in log-domain)
22        logR = log_phat - log_p
23        if log(rand()) < logR
24            w = wHat
25            log_p = log_phat
26            nAccept += 1
27            @printf("Accepted sample %d, acceptance rate = %f\n",s,nAccept/s)
28        end
29        samples[s,:] = w
30    end
31
32    return samples
33 end
```

Figure 13: Metropolis with modified Gaussian noise standard deviation.

²The “positive” variables are some of the positive weights when you fit an L2-regularized logistic regression model to the this data. The “negative” variables are some of the negative regression weights in that model, and the “neutral” ones are set to 0 in that model.

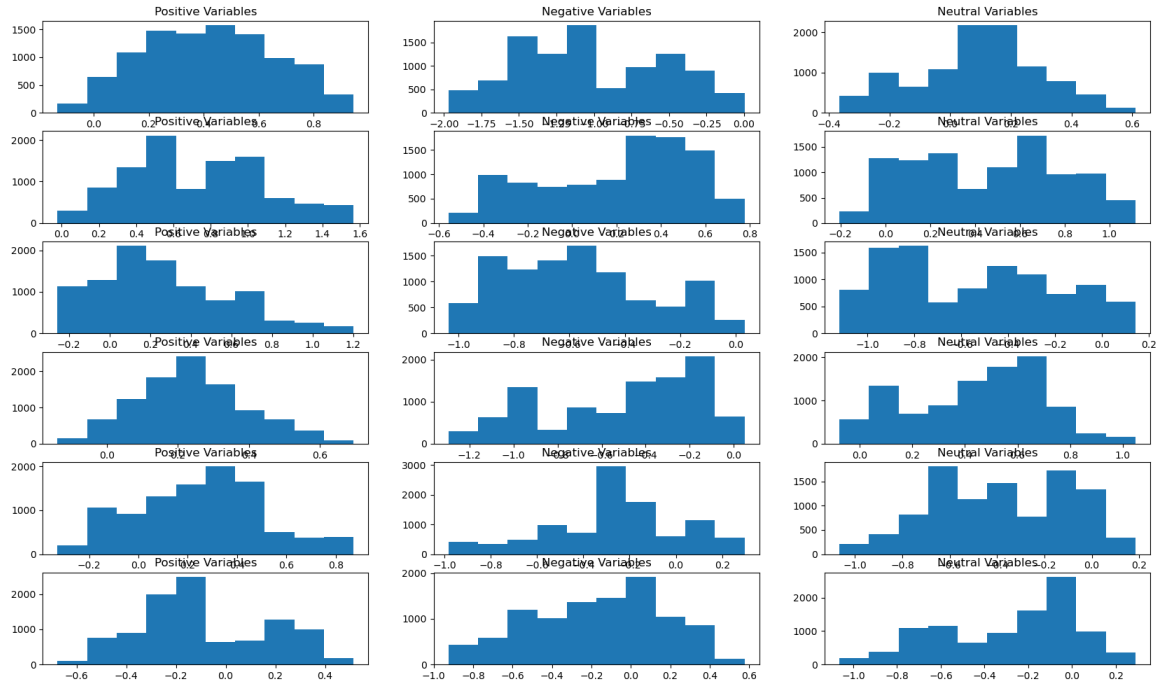


Figure 14: Posterior histogram with modified Gaussian noise.

3. Modify the proposal to use $\hat{w} \sim \mathcal{N}(w, (1/10000)I)$. Do you think this performs better or worse than the previous choice? (Briefly explain.)

Answer: Worse. I think by using such a low noise variance, we are being too conservative. In other words, our step size is too small and convergence would take much longer to be achieved. For example, in part 1, our noise was too high and we were far from the desired stationary distribution, causing all samples to be ignored. in part 2, we took the middle ground and used a moderate variance, causing a good proportion of samples to be accepted at a reasonable convergence rate. By using too low of a variance, it takes a long time to converge to a stationary distribution representing the posterior.

2.3 D-Separation

Consider the DAG model below, for distinguishing between different causes of shortness-of-breath (dyspnoea) and the causes of an abnormal lung x-ray, while modelling potential causes of these diseases too (whether the person is a smoker or had a ‘visit’ to a country with a high degree to tuberculosis).

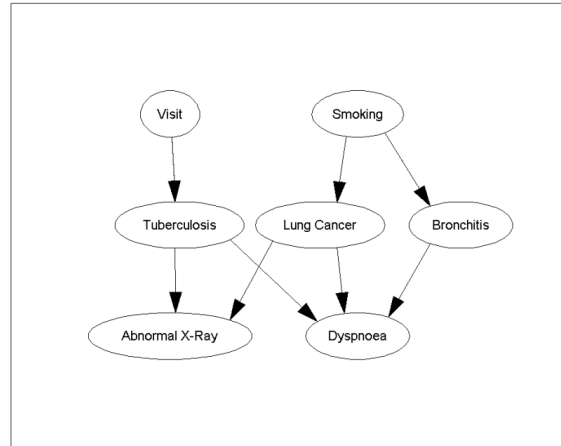


Figure 15: BayesNet DAG.

We'll assume that the distribution over the variables is “faithful” to the graph, meaning that variables are conditionally independent if and only if the graph structure implies their independence. Under this assumption, **say why each of the following conditional independence statements is true or false (provide a very-brief justification):**

1. $(\text{Smoking}) \perp (\text{Bronchitis})$.

Answer: False. Smoking is the parent of Bronchitis and a direct path between them exists. (case 1)

2. $(\text{Smoking}) \perp (\text{Dyspnoea})$.

Answer: False. An unblocked path through both Bronchitis and Lung Cancer exists between Smoking and Dyspnoea. (case 1)

3. $(\text{Tuberculosis}) \perp (\text{Lung Cancer})$.

Answer: True. These share two children, but the children are not observed, making them independent (case 3).

4. $(\text{Abnormal X-Ray}) \perp (\text{Dyspnoea})$.

Answer: False. These share parents that are not observed making them dependent (case 2).

5. $(\text{Abnormal X-Ray}) \perp (\text{Visit}) \mid (\text{Tuberculosis})$.

Answer: True. The path between these two is blocked since Tuberculosis is observed (case 1).

6. $(\text{Bronchitis}) \perp (\text{Lung Cancer}) \mid (\text{Smoking})$.

Answer: True. These two have a common parent, which is observed, making these independent (case 2).

7. $(\text{Tuberculosis}) \perp (\text{Lung Cancer}) \mid (\text{Dyspnoea})$.

Answer: False. These two have a common children that is observed, and are therefore dependent (case 3).

8. $(\text{Visit}, \text{Smoking}) \perp (\text{Abnormal X-Ray}, \text{Dyspnoea}) \mid (\text{Tuberculosis}, \text{Lung Cancer}, \text{Bronchitis})$

Answer: True. Aggregating these joint nodes into single nodes, we see that the chain between them is blocked. (case 1)

9. $(\text{Smoking}) \perp (\text{Visit}) \mid (\text{Dyspnoea})$.

Answer: False. Dyspnoea is an observed grandchild of these, and therefore, they are dependent (case 3).

10. $(\text{Tuberculosis}) \perp (\text{Bronchitis}) \mid (\text{Abnormal X-Ray})$.

Answer: False. Abnormal X-Ray is observed, so Tuberculosis and Lung cancer are dependent (case 3), lung cancer is dependent on Bronchitis (case 2) so Tuberculosis and Bronchitis are dependent.

3 Mixture Models

Cancelled.

4 Very-Short Answer Questions

1. What is the relationship between using a product of Gaussians and using a multivariate Gaussian?

Answer: Product of Gaussians is a subset of Multivariate Gaussians where the covariance matrix is diagonal, indicating independence among the Gaussian components.

2. How does the affine property allow us to sample from multivariate Gaussians?

Answer: Using the affine property, we can generate samples from independent standard normal distributions and transform these samples to achieve the desired multivariate Gaussian.

3. With a Gaussian likelihood, a Gaussian prior for its mean, and a fixed covariance, how do we know that the posterior predictive will also be a Gaussian?

Answer: Posterior is proportionate to product of the likelihood and the prior, both of which are Gaussians. Therefore, using the "Product of Gaussian Densities" property, the posterior is also a Gaussian.

4. How do the sparsity patterns of the covariance and precision matrix in a multivariate Gaussian relate to independence and conditional independence in the models?

Answer: The off-diagonal sparsity of the covariance matrix shows the independence among components of the model. The sparsity of the precision matrix can be used along with d-separation rules to find conditional independence among variables.

5. In linear discriminant analysis, why might we not assign a point to its closest mean?

Answer: This can happen when the class proportions are not equal, that is when the dataset is unbalanced.

6. The maximum of the posterior predictive in Bayesian linear regression gives the same prediction as if we used the MAP estimate. What is a reason we might use Bayesian linear regression anyways?

Answer: Bayesian linear regression also provides us with "error bars" or indicators of the model's confidence on its predictions, which is critical in certain applications.

7. What is the key difference between Monte Carlo approximations and variational approximations?

Answer: In Monte Carlo approximations we turn inference into sampling, whereas, in variational methods we perform inference through optimization methods.

8. What is a key advantage of "end-to-end" training of computer vision system?

Answer: The ability to optimize using backpropagation and stochastic gradient is a key advantage of end-to-end systems.

9. What are your 3 favourite properties of the exponential family?

Answer:

- They have simple and intuitive close-form MLEs
- In their canonical form, they are guaranteed to have conjugate priors.
- The NLL is convex

10. What is the difference between computing marginals and computing the stationary distribution of a Markov chain.

Answer: marginalization is the probability of being in a certain state at a certain time, while stationary distribution is the probability of being in a certain state at time infinity. Additionally, stationary distribution may not exist for all markov chains where marginalization can be performed for all.

11. What is the cost of generating a sample from a Markov chain of length d with k possible states for each time? What is the cost of decoding?

Answer: The cost of generating a sample is $O(kd)$ with ancestral sampling since for each time step we have to make $k-1$ comparisons if we use inverse transform sampling. Using Viterbi decoding, the cost of decoding is $O(dk^2)$.

12. In what setting is it unnecessary to include the q function in the Metropolis-Hastings acceptance probability?

Answer: For symmetric proposal distributions q , where $q(\hat{x} | x) = q(x | \hat{x})$ since they cancel out in the formula.

13. What is “explaining away”?

Answer: It is the case where multiple nodes share a child, and knowing (conditioning) one of these parents make the probability of other parents less/more likely.

14. If two variables are not d-separated, are they necessarily dependent? If two variables are d-separated, are they necessarily independent?

Answer: If two variables are d-separated, then they are necessarily independent. However, if two variables are not d-separated, it does not mean that they are necessarily dependent.

15. Describe how we could use ancestral sampling to sample from the joint density over x and y defined by a Gaussian discriminant analysis model.

Answer: in GDA, we use the product rule to change the joint probability into the product of $p(y)$ and $p(x|y)$. With ancestral sampling, we first sample a class c for y using $p(y)$, then produce a sample from $p(x|y = c)$.

GPU Access

We applied for UBC to fund some GPUs (RTX 3070 8GB) and a lot of RAM (192 GB) for use by students for their coursework. This is the first semester that these resources will be used, so there may be a learning curve to use these (from your side and our side). If you would like to use these for your course project, please make a private post on Piazza with the following information:

1. Names of people on the project.
2. Estimated number of GPU/CPU hours.
3. Roughly how much memory your jobs will need.
4. Roughly how much disk usage you will need.
5. Are there datasets you plan to use that other groups might also be using?