CPSC 340 Assignment 1 (due Friday September 16 at 11:55pm)

Commentary on Assignment 1: CPSC 340 is tough because it combines knowledge and skills across several disciplines. To succeed in the course, you will need to know or very quickly get up to speed on:

- Math to the level of the course prerequisites: linear algebra, multivariable calculus, some probability.
- Basic Julia programming, and the ability to translate from math to programming and back.
- Statistics, algorithms, and data structures to the level of the course prerequisites.
- Some basic LaTeX skills so that you can typeset equations and submit your assignments.

The purpose of this assignment is to make sure you are prepared for this course. We anticipate that each of you will have different strengths and weaknesses, so don't be worried if you struggle with *some* aspects of the assignment. But if you find this assignment to be very difficult overall, that is a sign that you may not be prepared to take CPSC 340 at this time. Future assignments will be more difficult than this one (and probably around the same length).

Questions 1-4 are on review material, that we expect you to know coming into the course. The rest is related to the first few lectures.

IMPORTANT!!!!! Before proceeding, please carefully read the homework instructions posted on Piazza.

You may receive a 50% deduction on the assignment if you don't follow these instructions.

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

You may also want to read the answers to this Quora question as motivation:
`https://www.quora.com/Why-should-one-learn-machine-learning-from-scratch-rather-than-just-learning-to-use-the-available-libraries`

Basic Information

1. Name:

   Answer: Masoud Mokhtari

2. Student ID:

   Answer: 14186167

   Answer: Using two late days for this assignment.

1 Linear Algebra Review

For these questions you may find it helpful to review these notes on linear algebra:
`http://www.cs.ubc.ca/~schmidtm/Documents/2009_Notes_LinearAlgebra.pdf`

1.1 Basic Operations

Use the definitions below,

$$\alpha = 2, \quad x = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}, \quad y = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}, \quad A = \begin{bmatrix} 3 & 2 & 2 \\ 1 & 3 & 1 \\ 1 & 1 & 3 \end{bmatrix}, \quad I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and use x_i to denote element i of vector x. Evaluate the following expresions, showing at least one intermediate step of work:

1. $\|x\|$ (Euclidean norm of x).

   Answer: $\|x\| = \sqrt{0^2 + 1^2 + 2^2} = \sqrt{5}$

2. $\alpha(x + y)$ (vector addition and scalar multiplication).

   Answer: $\alpha(x+y) = 2 \times \begin{bmatrix} 0+3 \\ 1+4 \\ 2+5 \end{bmatrix} = \begin{bmatrix} 2\times 3 \\ 2 \times 5 \\ 2 \times 7 \end{bmatrix} = \begin{bmatrix} 6 \\ 10 \\ 14 \end{bmatrix}$

3. $x^T y = \sum_{i=1}^n x_i y_i$ (inner product).

   Answer: $x^T y = \sum_{i=1}^n x_i y_i = 0 \times 3 + 1 \times 4 + 2 \times 5 = 14$

4. xy^T (outer product).

   Answer: $xy^T = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} \times \begin{bmatrix} 3 & 4 & 5 \end{bmatrix} = \begin{bmatrix} 0\times 3 & 0 \times 4 & 0 \times 5 \\ 1 \times 3 & 1 \times 4 & 1 \times 5 \\ 2 \times 3 & 2 \times 4 & 2 \times 5 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 3 & 4 & 5 \\ 6 & 8 & 10 \end{bmatrix}$

5. Ax (matrix-vector multiplication).

   Answer: $Ax = \begin{bmatrix} 3 & 2 & 2 \\ 1 & 3 & 1 \\ 1 & 1 & 3 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 3\times 0 + 2 \times 1 + 2 \times 2 \\ 1 \times 0 + 3 \times 1 + 1 \times 2 \\ 1 \times 0 + 1 \times 1 + 3 \times 2 \end{bmatrix} = \begin{bmatrix} 6 \\ 5 \\ 7 \end{bmatrix}$

6. $x^T Ax$ (quadratic form).

   Answer: $x^T Ax = \begin{bmatrix} 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} 3 & 2 & 2 \\ 1 & 3 & 1 \\ 1 & 1 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} 6 \\ 5 \\ 7 \end{bmatrix} = 19$

7. Solve for a vector v that satisfies $(I_3 - xx^T)v = y$ (linear system).

   Answer: $(I_3 - xx^T)v = y \rightarrow \begin{bmatrix} 1-0 & 0-0 & 0-0 \\ 0-0 & 1-1 & 0-2 \\ 0-0 & 0-2 & 1-4 \end{bmatrix} v = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} \rightarrow \begin{bmatrix} v_1 \\ -2v_3 \\ -2v_2 - 3v_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} \rightarrow$
   $v = \begin{bmatrix} 3 \\ \frac{1}{2} \\ -2 \end{bmatrix}$

1.2 Matrix Algebra Rules

Assume that $\{x, y, z\}$ are $n \times 1$ column vectors and $\{A, B, C\}$ are $n \times n$ real-valued matrices, 0 is the zero matrix of appropriate size, and I is the identity matrix of appropriate size. State whether each of the below is true in general (you do not need to show your work).

1. $x^T x = \|x\|^2$.

   Answer: True

2. $x^T x = x x^T$.

   Answer: False

3. $(x - y)^T (y - x) = \|x\|^2 - 2x^\top y + \|y\|^2$.

   Answer: False

4. $AB = BA$.

   Answer: False

5. $A(B + C) = AB + AC$.

   Answer: True

6. $(AB)^T = A^T B^T$.

   Answer: False

7. $x^T A y = y^T A^T x$.

   Answer: True

8. $A^n = (A^n)^T$ for any non-negative integer n if A is symmetric.

   Answer: True

9. $A^\top A = I$ if the columns of A are orthonormal.

   Answer: True

2 Probability Review

For these questions you may find it helpful to review these notes on probability:
`http://www.cs.ubc.ca/~schmidtm/Courses/Notes/probability.pdf`
And here are some slides giving visual representations of the ideas as well as some simple examples:
`http://www.cs.ubc.ca/~schmidtm/Courses/Notes/probabilitySlides.pdf`

2.1 Rules of probability

Answer the following questions. You do not need to show your work.

1. Consider two events A and B such that $\Pr(A, B) = 0$ (they are mutually exclusive). If $\Pr(A) = 0.4$ and $\Pr(A \cup B) = 0.95$, what is $\Pr(B)$? Note: $p(A, B)$ means "probability of A and B" while $p(A \cup B)$ means "probability of A or B". It may be helpful to draw a Venn diagram.

   Answer: $Pr(B) = 0.55$

2. Instead of assuming that A and B are mutually exclusive ($\Pr(A, B) = 0$), what is the answer to the previous question if we assume that A and B are independent?

   Answer: $Pr(B) = 0.9167$

3. You are offered the opportunity to play the following game: first your opponent rolls a 4 - sided dice and records the outcome r_1. Then you roll a $(5 - r_1)$ - sided dice and record the outcome r_2. Your payout is $r_1 + r_2 - 1$ dollars. You can enter the game either before or after your opponent's turn.

   - If you enter *after* your opponent's turn you know r_1. What is a fair price for a ticket in this case, i.e., what is the expected payout as a function of r_1?

     Answer: $\sum_{r_2=1}^{5-r_1}(r_1 + r_2 - 1) \times \frac{1}{5-r_1}$

   - If you enter *before* your opponent's turn you do not know r_1. What is the expected payout now?

     Answer: $\sum_{r_1=1}^{4}\sum_{r_2=1}^{5-r_1}(r_1 + r_2 - 1) \times \frac{1}{5-r_1} \times \frac{1}{4} = 3.25$

2.2 Bayes Rule and Conditional Probability

Answer the following questions. You do not need to show your work.

Suppose a drug test produces a positive result with probability 0.95 for drug users, $P(T = 1|D = 1) = 0.95$. It also produces a negative result with probability 0.99 for non-drug users, $P(T = 0|D = 0) = 0.99$. The probability that a random person uses the drug is 0.0001, so $P(D = 1) = 0.0001$.

1. What is the probability that a random person would test positive, $P(T = 1)$?

   Answer: $P(T = 1) = 0.010094$

2. In the above, do most of these positive tests come from true positives or from false positives?

   Answer: False positives

3. What is the probability that a random person who tests positive is a user, $P(D = 1|T = 1)$?

   Answer: 0.009411

4. Suppose you have given this test to a random person and it came back positive, are they likely to be a drug user?

   Answer: No not likely

5. What is one factor you could change to make this a more useful test?

   Answer: Given the high number of non-drug users, $P(T = 1|D = 0)$ should be minimized. That is, the number of false positives must be reduced by increasing test accuracy. If it's not possible to change the test procedure, one can increase the number of trials per patient instead.

3 Calculus Review

For these questions you may find it helpful to review these notes on calculus:
`http://www.cs.ubc.ca/~schmidtm/Courses/Notes/calculus.pdf`

3.1 One-variable derivatives

Answer the following questions. You do not need to show your work.

1. Find the derivative of the function $f(x) = 3x^2 - 2x + 5$.

   Answer: $\frac{df(x)}{dx} = 6x - 2$

2. Find the derivative of the function $f(x) = x^2 \cdot \exp(x)$.

   Answer: $\frac{df(x)}{dx} = 2x \cdot \exp(x) + x^2 \cdot \exp(x)$

3. Let $p(x) = \frac{1}{1+\exp(-x)}$ for $x \in \mathbb{R}$. Compute the derivative of the function $f(x) = x - \log(p(x))$ and simplify it by using the function $p(x)$.

   Answer: $p(x)$

Note that in this course we will use $\log(x)$ to mean the "natural" logarithm of x, so that $\log(\exp(1)) = 1$. Also, observe that $p(x) = 1 - p(-x)$ for the final part.

3.2 Multi-variable derivatives

Compute the gradient $\nabla f(x)$ of each of the following functions. You do not need to show your work.

1. $f(x) = x_1^2 + \exp(x_2)$ where $x \in \mathbb{R}^2$.

   Answer: $\begin{bmatrix} 2x_1 \\ \exp(x_2) \end{bmatrix}$

2. $f(x) = \exp(x_1 + x_2 x_3)$ where $x \in \mathbb{R}^3$.

   Answer: $\begin{bmatrix} \exp(x_1 + x_2 x_3) \\ \exp(x_1 + x_2 x_3) \cdot x_3 \\ \exp(x_1 + x_2 x_3) \cdot x_2 \end{bmatrix}$

3. $f(x) = a^T x$ where $x \in \mathbb{R}^2$ and $a \in \mathbb{R}^2$.

   Answer: $\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$

4. $f(x) = x^\top A x$ where $A = \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix}$ and $x \in \mathbb{R}^2$.

   Answer: $\begin{bmatrix} 4x_1 - 2x_2 \\ 2x_2 - 2x_1 \end{bmatrix}$

5. $f(x) = \frac{1}{2}\|x\|^2$ where $x \in \mathbb{R}^d$.

   Answer: $\begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_d \end{bmatrix}$

Hint: it is helpful to write out the linear algebra expressions in terms of summations.

3.3 Derivatives of code

For these questions you may find it helpful to review this list of useful Julia commands:
`http://www.cs.ubc.ca/~schmidtm/Courses/Notes/juliaCommands.txt`

The zip file `a1.zip` contains a Julia file named `grads.jl` which defines several functions that take in a vector as input. Complete the functions `grad1`, `grad2`, and `grad3` (which compute the gradients of `func1`, `func2`, and `func3`). Include the code in the PDF file for this section, and also in your zip file.

Hint: for many people it's easiest to first understand on paper what the code is doing, then compute the gradient, and then translate this gradient back into code. We've given you `func0` and `grad0` as an example. Also, we've provided the function `numGrad` which approximates the gradient numerically to help you debug. Below is an example of using these functions:

Note: do not worry about the distinction between row vectors and column vectors here. For example, if the correct answer is a vector of length 5, we'll accept vectors of size 5×1 or 1×5. In future assignments we will be more careful to always use column vectors.

Answer: The code for each function is provided in Figures 1, 2, and 3

```
10  ### Function 1
11  function func1(x)
12      f = 0;
13      n = length(x)
14      for i in 1:n
15          f += (1/2)*(x[i] - i)^2
16      end
17      return f
18  end
19
20  function grad1(x)
21      n = length(x);
22      g = zeros(n);
23      for i in 1:n
24          g[i] += x[i] - i
25      end
26      return g
27  end
```

Figure 1: Function 1 Gradient Implementation

```
### Function 2
using LinearAlgebra
func2(x) = dot(x,1:length(x))

function grad2(x)
    n = length(x);
    g = zeros(n);

    for i in 1:n
        g[i] = i
    end

    return g
end
```

Figure 2: Function 2 Gradient Implementation

```
### Function 3
func3(x) = sum(max.(x,0).^2)

function grad3(x)
    n = length(x);
    g = zeros(n);
    g[x .<= 0] .= 0
    g[x .> 0] = 2 * x[x .> 0]
    return g
end
```

Figure 3: Function 3 Gradient Implementation

4 Algorithms and Data Structures Review

For these questions you may find it helpful to review these notes on big-O notation:
`http://www.cs.ubc.ca/~schmidtm/Courses/Notes/bigO.pdf`

4.1 Trees

Answer the following questions You do not need to show your work.

1. What is the maximum number of *leaves* you could have in a binary tree of depth l?

   Answer: 2^l

2. What is the maximum number of *internal nodes* (excluding leaves) you could have in a binary tree of depth l?

   Answer: $2^l - 1$

Note: we'll use the standard convention that the leaves are not included in the depth, so a tree with depth 1 has 3 nodes with 2 leaves.

4.2 Common Runtimes

Answer the following questions using big-O notation You do not need to show your work.

1. What is the cost of running the mergesort algorithm to sort a list of n numbers?

   Answer: O(nlog(n))

2. What is the cost of finding the third-largest element of an unsorted list of n numbers?

   Answer: O(nlog(n))

3. What is the cost of finding the smallest element greater than 0 in a *sorted* list with n numbers?

   Answer: O(n)

4. What is the cost of finding the value associated with a key in a hash table with n numbers? (Assume the values and keys are both scalars.)

   Answer: Assuming no collisions: O(1) With collision (worst case): O(n)

5. What is the cost of computing the matrix-vector product Ax when A is $n \times d$ and x is $d \times 1$?

   Answer: $O(n \times d)$

6. What is the cost of computing the quadratic form $x^T Ax$ when A is $d \times d$ and x is $d \times 1$?

   Answer: $O(d^3)$

7. How does the answer to the previous question change if A has only z non-zeroes? (You can assume $z \geq d$)

   Answer: $O(z^2)$

4.3 Running times of code

Included in `a1.zip` is file named `bigO.jl`, which defines several functions that take an integer argument n. For each function, state the running time as a function of n, using big-O notation.

- func1: O(n)
- func2: O(n)
- func3: O(1)
- func4: $O(n^2)$
- func5: $O(n^3)$

5 Summary Statistics and Data Visualization

The file *a1.zip* contains estimates of the influenza-like illness percentage over 52 weeks on 2005-06 by Google Flu Trends in a comma-separated values (CSV) file. You can open this with Excel or other spreadsheet programs; the first row gives the abbreviation of the region names for each column, and each row gives the estimate for a week. After you change to the a0 directory, you can load this data in Julia using:

```
using DelimitedFiles
dataTable = readdlm("fluTrends.csv",',')
```

This creates an two-dimensional array of type "Any" populated with all the information in the CSV file.

5.1 Summary Statistics

Report the following statistics: the minimum, maximum, mean, median, and mode of all values across the dataset. In light of thea above, is the mode a reliable estimate of the most "common" value? Describe another way we could give a meaningful "mode" measurement for this (continuous) data.

Hint: Since the first row of the CSV file is just the names of the columns, we can create a matrix X containing the data stored as real numbers using:

```
X = real(dataTable[2:end,:])
```

You can make Julia dispaly the matrix X using

```
@show X
```

The *show* macro can be used to display the result of any expression, like showing the tenth row of X:

```
@show X[10,:]
```

Note that this can be run inside functions, so it's helpful for debugging.

Julia has a mean and median function available, if you include the Statistics package. This package does not have a mode command, so I've included one in 'misc.jl'.

Answer: The statistics are shown in Figure 4 and summarized below:

- minimum: 0.352
- maximum: 4.862
- mean: 1.325
- median: 1.160
- mode: 0.77

Mode is not a reliable estimate of the most common percentage here since we are dealing with continuous values and a small difference could cause two very close percentages to not be considered when computing the mode. In this case, visualizing the data distribution using a histogram plot allows the identification of the most common value. Another approach (with the assumption of normal distribution) is using the mean (and std). However, if outliers are present, mean and std could get skewed.

```
minimum(X) = 0.352
maximum(X) = 4.862
mean(X) = 1.324625
median(X) = 1.1589999999999998
mode(X) = 0.77
```

Figure 4: Statistics of the influenza-like dataset.

5.2 Data Visualization

Consider the figure below. The figure contains the following plots, in a shuffled order:

1. A histogram showing the distribution of all values in the matrix X.
2. A boxplot grouping data by weeks, showing the distribution across regions for each week.
3. A scatterplot between the two regions with highest correlation.
4. A single histogram showing the distribution of *each* column in X.
5. A scatterplot between the two regions with lowest correlation.
6. A plot containing the weeks on the x-axis and the percentages for each region on the y-axis.

Match the plots (labeled A-F) with the descriptions above (labeled 1-6), with an extremely brief (a few words is fine) explanation for each decision. Hint: you can generate similar plots by adding the Plots package. To add this package and make a plot, run the following from the Julia REPL:

```
using Pkg # Loads the package manager
Pkg.add("Plots") # Only needs to be done once  (installs a Julia-callable Python build)
using Plots # Do this once per session
plot(1:52,X[:,1]) # Plot the first row
```

To generate similar-looking plots you can use the functions 'plot', 'scatter', 'histogram', and 'boxplot' (which is from the StatsPlot package).

Answer:

- $C \rightarrow 1$: It shows the percentage distribution (frequency) of a single set.
- $B \rightarrow 2$: We see the mean, std and quartet stats of illness percentage for each week
- $F \rightarrow 3$: Higher correlation leads to a more linear (positive slope) plot
- $D \rightarrow 4$: Similar to C but with separate plots for each region
- $E \rightarrow 5$: Lower correlation is seen by the lower slop of the trendline.
- $A \rightarrow 6$: Well, weeks on the x-axis and percentage of each region on the y-axis.

5.3 Decision Surfaces

Consider the figure below, which plots a set of two-dimensional training examples and the decision surface produced by a "neural network" classifier (a model we'll see later in the course). How many training examples has the neural network mis-classified? (This figure is best viewed in colour.)

Answer: As shown in Figure 5, there are 17 mis-classifications.

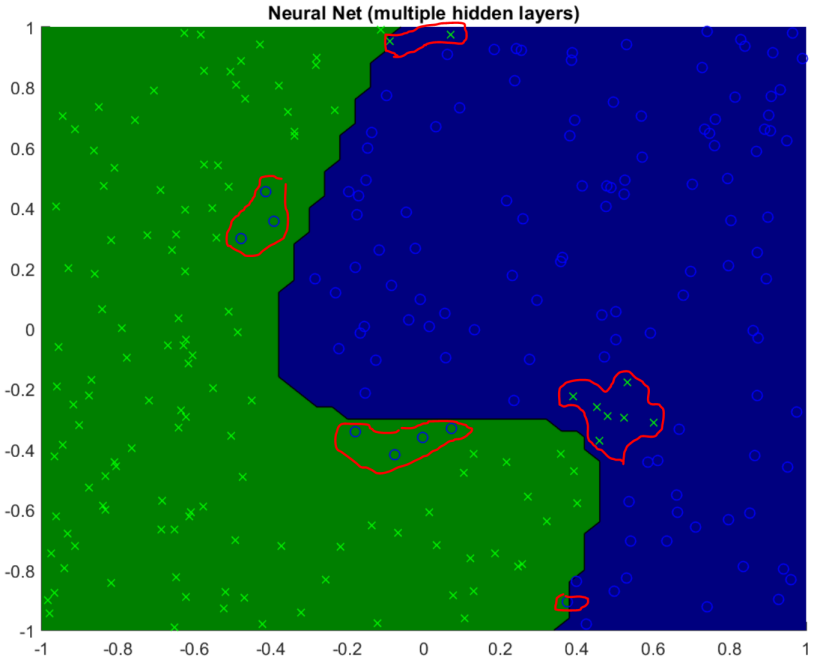


Figure 5: Mis-classified samples

6 Decision Trees

If you run the file *example_decisionStump.jl*, it will load a dataset containing longtitude and latitude data for 400 cities in the US, along with a class label indicating whether they were a "red" state or a "blue" state in the 2012 election.[1] Specifically, the first column of the variable X contains the longitude and the second variable contains the latitude, while the variable y is set to 1 for blue states and 2 for red states. After it loads the data, it plots the data and then fits two simple classifiers: a classifier that always predicts the most common label (1 in this case) and a decision stump that discretizes the features (by rounding to the nearest integer) and then finds the best equality-based rule (i.e., check if a feature is *equal* to some value). It reports the training error with these two classifiers, then plots the decision areas made by the decision stump. The plot should look like this:

Note that these functions use the "JLD" package for loading the data and the "Plots" package to do the plotting. You can install these packages using:

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

6.1 Equality vs. Inequality Splitting Rules

In class we discussed splitting rules based on inequalities rather than equalities. Is there a type of feature where it makes sense to use an equality-based splitting rule?

Answer: Yes, in the case where the feature is binary (True/False, 0/1), it does make sense to use an equality approach. In fact, in such cases, it may not make sense to use an inequality approach.

[1] The cities data was sampled from `http://simplemaps.com/static/demos/resources/us-cities/cities.csv`. The election information was collected from Wikipedia.

6.2 Decision Stump Implementation

The file *decisionstump.jl* contains a function that finds the best decision stump using the equality rule ("decisionStumpEquality"), and then returns a function that can apply this decision stump to new data. Instead of discretizing the data and using a rule based on testing an equality for a single feature, we want to check whether a feature is above a threshold and split the data accordingly (this is the more sane approach, which we discussed in class). Add a new function "decisionStump" to *decision_stump.jl* that finds the best inequality-based rule, and report the updated error you obtain by using inequalities instead of discretizing and testing equality.

Hint: you may want to start by copy/pasting the contents of of the "decisionStumpEquality" function and then make modifications from there. Note that you should remove the calls to the "round" function for the inequality case. Make sure that you maintain the same input/output format in your function, since otherwise subsequent questions will not work (it should produce a plot that divides the US into a northern blue and a southern red area). If you are new to Julia, you may also want to look at *majorityPredictor.jl* to get an idea of the syntax in a simpler case.

Answer: The obtained error is 0.25, and the implementation is provided in Figure 6.

```
# Get the size of the data matrix
(n,d) = size(X)

# Initialize the "best rule" with the baseline rule (no split)
y_mode = mode(y)
minError = sum(y .!= y_mode);
splitVariable = [];
splitValue = [];
splitYes = y_mode;
splitNo = [];

# Search for the best rule
# (Uses O(n^2d) approach to keep code simple)
yhat = zeros(n)
for j in 1:d
    # Try unique values of column as split values
    for val in unique(X[:,j])

        # Test whether each object satisfies equality
        yes = X[:,j] .>= val

        # Find correct label on both sides of split
        y_yes = mode(y[yes])
        y_no = mode(y[.!yes])

        # Make predictions
        yhat[yes] .= y_yes
        yhat[.!yes] .= y_no

        # Compute error
        trainError = sum(yhat .!= y)

        # Update best rule
        if trainError < minError
            minError = trainError
            splitVariable = j
            splitValue = val
            splitYes = y_yes
            splitNo = y_no
        end
    end
end

# Now that we have the best rule,
# let's build our splitting function
function split(Xhat)
    (t,d) = size(Xhat)
    if isempty(splitVariable)
        return fill(true,t)
    else
        return (Xhat[:,splitVariable] .>= splitValue)
    end
end
```

Figure 6: Inequality-based implementation of the decision stump.

6.3 Decision Tree to Program

Once your *decisionStump* function is finished, the script *example_decisionTree* will be able to fit a decision tree of depth 2 to the same dataset (which results in a lower training error). Look at how the decision tree is stored and how the (recursive) *predict* function works. Using the same splits as the fitted depth-2 decision tree, write out what an alternate version of the predict function would be for classifying one training example as a simple program using if/else statements (as in the first slide of L3 that has the title "Decision Trees").

Hint: you can use use the "@show" macro to print the values of various expressions during the execution of a .jl file.

Answer: As shown in Figure 7, using the "@show" macro, we find the information required to re-implement the decision tree. The code is provided in Figure 8, and the achieved results (which match the original results) are provided in Figure 9.

```
splitValue = 37.695206
splitYes = 1
splitNo = 2
splitVariable = 1
splitValue = -96.032692
splitYes = 1
splitNo = 2
splitVariable = 1
splitValue = -112.548331
splitYes = 2
splitNo = 1
```

Figure 7: Original depth two decision tree parameters

```
function predict(Xhat)
    (t,d) = size(Xhat)
    yhat = zeros(t)

    for sample_idx in 1:t
        if Xhat[sample_idx, 2] >= 37.695206
            yhat[sample_idx] = 1
        else
            if Xhat[sample_idx, 1] >= -96.032692
                yhat[sample_idx] = 1
            else
                if Xhat[sample_idx, 1] >= -112.548331
                    yhat[sample_idx] = 2
                else
                    yhat[sample_idx] = 1
                end
            end
        end
    end

    return yhat
end
```

Figure 8: Simplified implementation of the decision tree

```
yhat = [2.0, 2.0, 1.0, 2.0, 2.0, 2.0, 2.0, 1.0, 2.0, 2.0, 2.0, 2.0, 1.0, 1.0, 2.0, 2.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 2.0, 1.0, 2.0, 2.0, 2.0, 2.0, 1.0, 1
.0, 1.0, 2.0, 1.0, 2.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 2.0, 2.0, 2.0, 2.0, 2.0, 1.0, 2.0, 2.0, 1.0, 2.0, 1.0, 2.0, 1.0, 2.0, 1.0, 1.0, 2.0, 2.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0,
2.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 2.0, 1.0, 1.0, 1.0, 2.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0,
 1.0, 2.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 2.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 2.0, 2.0
, 2.0, 1.0, 2.0, 2.0, 1.0, 1.0, 2.0, 1.0, 2.0, 1.0, 2.0, 2.0, 1.0, 2.0, 2.0, 2.0, 1.0, 2.0, 2.0, 1.0, 1.0, 1.0, 2.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 2.0, 1.
0, 1.0, 1.0, 2.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 2.0, 2.0, 2.0, 2.0, 1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 2.0, 1.0, 2.0, 1
.0, 1.0, 1.0, 1.0, 2.0, 2.0, 1.0, 1.0, 1.0, 2.0, 2.0, 2.0, 1.0, 2.0, 1.0, 2.0, 2.0, 2.0, 1.0, 2.0, 1.0, 1.0, 2.0, 1.0, 1.0, 2.0, 1.0, 2.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 1.0, 1.0, 2.0, 2.0, 1.0, 2.0, 2.0, 2.0, 1.0, 2.0, 1.0, 2.0, 1.0, 2.0, 2.0, 2.0, 1.0, 1.0, 2.0, 2.0, 2.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0,
 2.0, 2.0, 1.0, 1.0, 2.0, 2.0, 1.0, 2.0, 2.0, 1.0, 2.0, 1.0, 1.0, 1.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 2.0, 2.0, 2.0, 1.0, 2.0, 1.0, 2.0, 1.0, 2.0
, 2.0, 2.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 2.0, 1.0, 2.0, 1.0, 2.0, 1.0, 2.0, 1.0, 2.0, 2.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 2.0, 2.0, 2.0, 2.0, 1.0, 2.0, 1.0, 1.0, 1.0, 2.0, 1.
0, 2.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 2.0, 1.0, 2.0, 1.0, 2.0, 1.0, 2.0, 2.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 2.0, 1.0, 2.0, 2.0, 1.0, 2.0, 2.0, 1
.0, 2.0, 2.0, 1.0, 1.0, 2.0, 2.0, 1.0, 2.0, 2.0, 2.0, 2.0, 2.0, 1.0, 2.0, 2.0, 1.0, 2.0, 1.0]
Error with depth-2 decision tree: 0.172
```

Figure 9: Results of the simplified decision tree

6.4 Constructing Decision Trees

Report the training error generated by fitting a decision tree of depth 3, and hand in the corresponding decision surface plot.

Answer: A training error of 0.133 is obtained using a depth 3 decision tree. The resulting decision surface is shown in Figure 10.

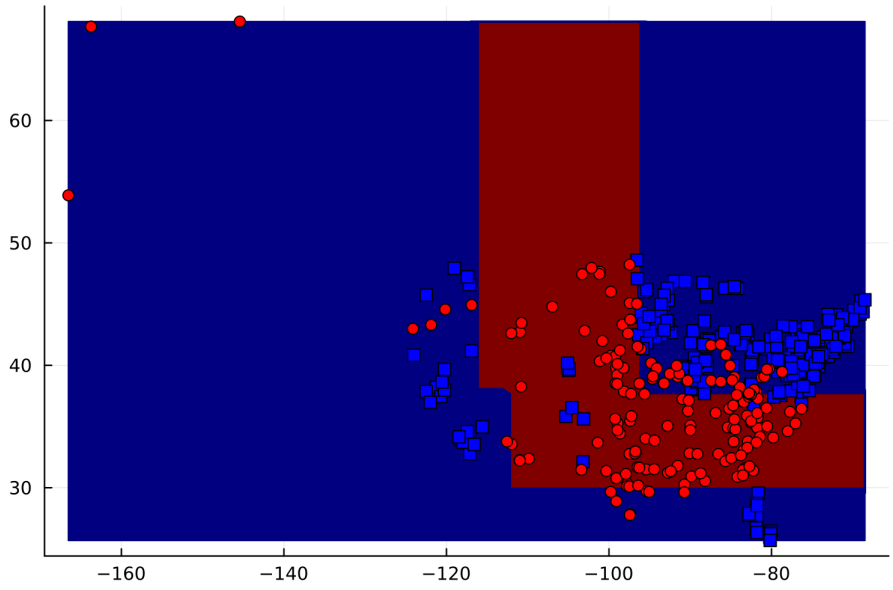


Figure 10: Decision surface of the depth 3 decision tree

HAVE YOU DOUBLE CHECKED THAT YOU'RE FOLLOWING ALL THE ASSIGNMENT SUBMISSION INSTRUCTIONS POSTED ON PIAZZA???

CPSC 340 Assignment 2 (due Friday September 30 at 11:55pm)

The assignment instructions are the same as Assignment 1, except you have the option to work in a group of 2. It is recommended that you work in groups as the assignment is quite long, but please only submit one assignment for the group.

Name(s) and Student ID(s): Masoud Mokhtari 14186167

1 Training and Testing

1.1 Training Error

Running `example_train.jl` fits decision trees of different depths using two different implementations: the "decisionStump" function from Assignment 1, and using a variant using a more sophisticated splitting criterion called the information gain. Describe what you observe. Can you explain the results?

Answer: Observations:

- We see that while the model using the "decisionStump" gets stuck at a training error of 0.11, the information gain-based model achieves a training error of 0 after only 8 layers.
- We see that the model's error when using lower than 5 layers is lower for the "decisionStump" compared to the information gain-based model.

Answer: Explanations:

- The information gain model manages to achieve a training error of 0 because it relies on minimizing entropy after each layer. Even though this could prevent the model from achieving maximum accuracy at the initial layers, it makes it more tractable for deeper layers to find an optimal split. This is why the model achieves an entropy of 0 where each leaf contains homogeneous samples, resulting in a training error of 0.
- The accuracy-based model independently maximizes the achieved accuracy at each layer without considering how such choices affect subsequent layers. This could cause a scenario (similar to the 2-feature example in the lecture) where no optimal split can be achieved.
- In this example, the accuracy-based model achieves higher accuracy at lower depths due to its focus on accuracy maximization, which is only useful in the first few layers.

1.2 Training and Testing Error Curves

Notice that the *citiesSmall.mat* file also contains test data, "Xtest" and "ytest". Running *example_trainTest* trains a depth-2 decision tree and evaluates its performance on the test data. With a depth-2 decision tree, the training and test error are fairly close, so the model hasn't overfit much.

Make a plot that contains the training error and testing error as you vary the depth from 1 through 15. How do each of these errors change with the decision tree depth?

Note: use the provided infogain-based decision tree code from the previous subsection.

Answer: The plot is shown in Figure 1, and the implementation to produce the plot is shown in Figure 2. We see that the training error keeps reducing until the model uses a depth of 9 where the training error becomes 0. For the test error, we see that it is generally decreasing up to depth 8. However, after that, the test error slightly increases and remains at around 0.095. This is most probably due to the model over-fitting the training dataset.

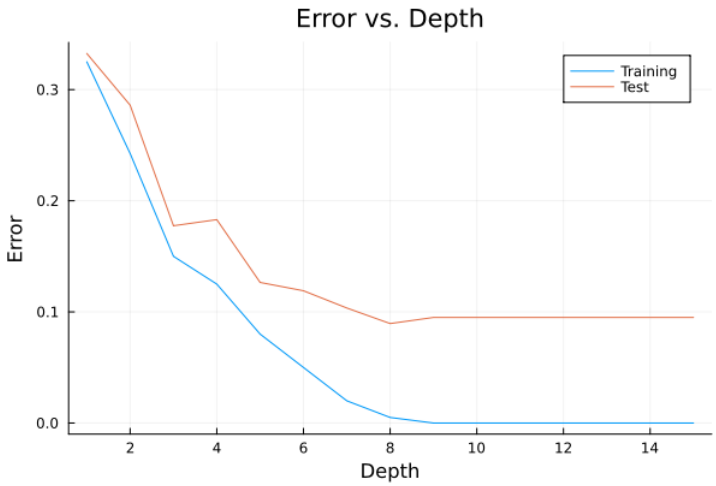


Figure 1: Training and test error vs. depth of decision tree.

```julia
using Printf

# Load X and y variable
using JLD
X = load("citiesSmall.jld","X")
y = load("citiesSmall.jld","y")
n = size(X,1)

trainErrors = zeros(15)
testErrors = zeros(15)
depths = 1:15

# Train a depth-2 decision tree
for depth in depths
    include("decisionTree_infoGain.jl")
    model = decisionTree_infoGain(X,y,depth)

    # Evaluate the training error
    yhat = model.predict(X)
    trainError = sum(yhat .!= y)/n
    @printf("Train error with depth-%d decision tree: %.3f\n",depth,trainError)
    trainErrors[depth] = trainError

    # Evaluate the test error
    Xtest = load("citiesSmall.jld","Xtest")
    ytest = load("citiesSmall.jld","ytest")
    t = size(Xtest,1)
    yhat = model.predict(Xtest)
    testError = sum(yhat .!= ytest)/t
    @printf("Test error with depth-%d decision tree: %.3f\n",depth,testError)
    testErrors[depth] = testError
end

using Plots
plot(depths, hcat(trainErrors, testErrors), title="Error vs. Depth", label["Training, Test"])
xlabel!("Depth")
ylabel!("Error")
```

Figure 2: Julia implementation to produce the plot in Figure 1

1.3 Validation Set

Suppose we're in the typical case where we don't have the labels for the test data. In this case, we might instead use a *validation* set. Split the training set into two equal-sized parts: use the first $n/2$ examples as a training set and the second $n/2$ examples as a validation set (we're assuming that the examples are already in a random order). What depth of decision tree would we pick if we minimized the validation set error? Does the answer change if you switch the training and validation set? How could we use more of our data to estimate the depth more reliably?

Note: use the provided infogain-based decision tree code from the previous subsection.

Answer: As shown in Figure 3, using this validation set, we would choose a depth of 3 since after this depth, the validation error increases. if we switch the validation and training sets, we achieve the plot shown in Figure 4, which indicates that we should choose a model with the depth of 6. We see that our choice of the validation set is not very reliable. A better way to approach this is to use k-fold validation, where the data is split into k folds, and one fold is selected as the validation set for each session. At the end, the validation errors are averaged to find a more reliable estimate for model selection.

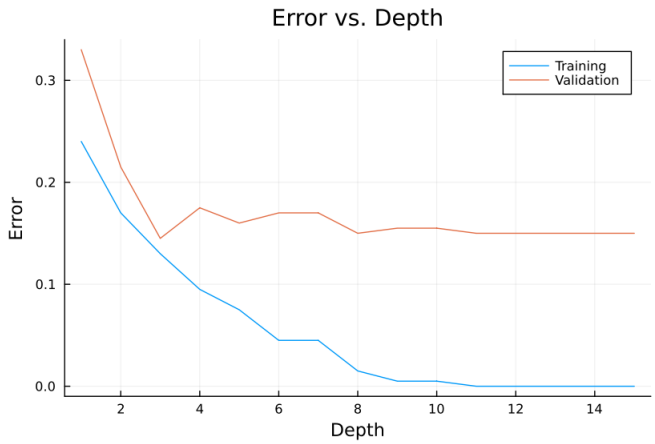


Figure 3: Validation and training error vs. depth for the original validation split

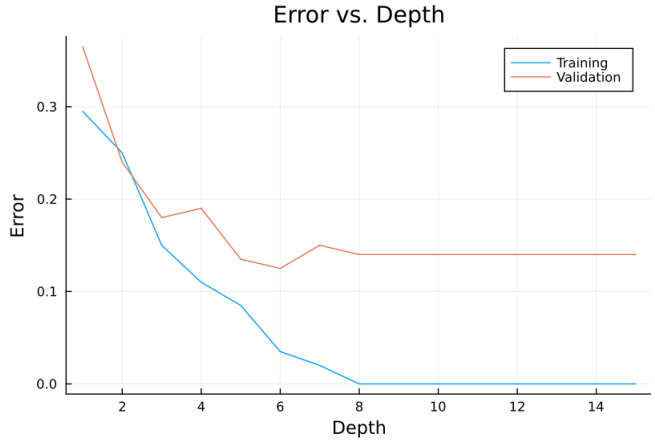


Figure 4: Validation and training error vs. depth for the reversed validation split

2 Naive Bayes

In this section we'll implement naive Bayes, a very fast classification method that is often surprisingly accurate for text data with simple representations like bag of words.

2.1 Naive Bayes by Hand

Consider the dataset below, which has 12 training examples and 3 features:

$$X = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} \text{spam} \\ \text{spam} \\ \text{spam} \\ \text{spam} \\ \text{spam} \\ \text{spam} \\ \text{spam} \\ \text{not spam} \\ \text{not spam} \\ \text{not spam} \\ \text{not spam} \\ \text{not spam} \end{bmatrix}.$$

The feature in the first column is <your name> (whether the e-mail contained your name), in the second column is "pharmaceutical" (whether the e-mail contained this word), and the third column is "PayPal" (whether the e-mail contained this word). Suppose you believe that a naive Bayes model would be appropriate for this dataset, and you want to classify the following test example:

$$\hat{x} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}.$$

2.1.1 Prior probabilities

Compute the estimates of the class prior probabilities (you don't need to show any work):

- $p(\text{spam})$.

  Answer: $p(\text{spam}) = \frac{7}{12}$

- $p(\text{not spam})$.

  Answer: $p(\text{not spam}) = \frac{5}{12}$

2.1.2 Conditional probabilities

Compute the estimates of the 6 conditional probabilities required by naive Bayes for this example (you don't need to show any work):

- $p(\text{<your name>} = 1 \mid \text{spam})$.

  Answer: $p(\text{<your name>} = 1 \mid \text{spam}) = \frac{1}{7}$

- $p(\text{pharmaceutical} = 0 \mid \text{spam})$.

  Answer: $p(\text{pharmaceutical} = 0 \mid \text{spam}) = \frac{1}{7}$

- $p(\text{PayPal} = 1 \mid \text{spam})$.

  Answer: $p(\text{PayPal} = 1 \mid \text{spam}) = \frac{4}{7}$

- $p(\text{<your name>} = 1 \mid \text{not spam})$.

  Answer: $p(\text{<your name>} = 1 \mid \text{not spam}) = \frac{4}{5}$

- $p(\text{pharmaceutical} = 0 \mid \text{not spam})$.

  Answer: $p(\text{pharmaceutical} = 0 \mid \text{not spam}) = \frac{3}{5}$

- $p(\text{PayPal} = 1 \mid \text{not spam})$.

  Answer: $p(\text{PayPal} = 1 \mid \text{not spam}) = \frac{1}{5}$

2.1.3 Prediction

Under the naive Bayes model and your estimates of the above probabilities, what is the most likely label for the test example? (Show your work.)

Answer:

- $p(\hat{y} = \text{"spam"} \mid \hat{x}) \approx p(\text{<your name>} = 1 \mid \hat{y} = \text{"spam"})p(\text{pharmaceutical} = 0 \mid \hat{y} = \text{"spam"})p(\text{PayPal} = 1 \mid \hat{y} = \text{"spam"}) * p(y = \text{"spam"}) = \frac{1}{7} \times \frac{1}{7} \times \frac{4}{7} \times \frac{7}{12} = 0.68\%$
- $p(\hat{y} = \text{"not spam"} \mid \hat{x}) \approx p(\text{<your name>} = 1 \mid \hat{y} = \text{"not spam"})p(\text{pharmaceutical} = 0 \mid \hat{y} = \text{"not spam"})p(\text{PayPal} = 1 \mid \hat{y} = \text{"not spam"}) \times p(y = \text{"not spam"}) = \frac{4}{5} \times \frac{3}{5} \times \frac{1}{5} \times \frac{5}{12} = 4\%$

Therefore, because $p(\hat{y} = \text{"spam"} \mid \hat{x}) < p(\hat{y} = \text{"not spam"} \mid \hat{x})$, the most likely label for our test sample is "not spam".

2.2 Bag of Words

If you run the script *example_BagOfWods.jl*, it will load the following dataset:

1. X: A sparse binary matrix. Each row corresponds to a newsgroup post, and each column corresponds to whether a particular word was used in the post. A value of 1 means that the word occured in the post.
2. *wordlist*: The set of words that correspond to each column.
3. y: A vector with values 1 through 4, with the value corresponding to the newsgroup that the post came from.
4. *groupnames*: The names of the four newsgroups.
5. $Xtest$ and $ytest$: the word lists and newsgroup labels for additional newsgroup posts.

Answer the following:

1. Which word is present in the newsgroup post if there is a 1 in column 50 of X?

   Answer: The 50th element of *wordlist* is "league".

2. Which words are present in training example 500?

   Answer:

   - car
   - engine
   - evidence
   - problem
   - system

3. Which newsgroup name does training example 500 come from?

   Answer: The newsgroup is from "rec.*"

   Answer: The answers above were extracted by modifying the code as shown in Figure 5

```
The 50th word in the wordlist is:
league

The following words are present in example 500:
car
engine
evidence
problem
system

Example 500 comes from the following newsgroup:
rec.*
```

Figure 5: Obtained information from the bag of words representation.

2.3 Naive Bayes Implementation

If you run the function *example_decisionTree_newsgroups.jl* it will load the newsgroups dataset and report the test error for decision trees of different sizes (it may take a while for the deeper trees, as this is a sub-optimal implementation). On other other hand, if you run the function *example_naiveBayes.jl* it will fit the basic naive Bayes model and report the test error.

While the *predict* function of the naive Bayes classifier is already implemented, the calculation of the variable *p_xy* is incorrect (right now, it just sets all values to 1/2). Modify this function so that *p_xy* correctly computes the conditional probabilities of these values based on the frequencies in the data set. Hand in your code and report the test error that you obtain.

Answer: The modified portion of the code is shown in Figure 6, and the achieved test error is 0.188, which is lower than even a depth 20 decision tree.

```
18  # We will store p(x(i,j) = 1 | y(i) = c) in p_xy(1,j,c)
19  # We will store p(x(i,j) = 0 | y(i) = c) in p_xy(2,j,c)
20  p_xy = (1/2)ones(2,d,k)
21  for j in 1:d
22    for c in 1:k
23        p_xy[1, j, c] = sum(X[y.==c,j].==1)/sum(y.==c)
24        p_xy[2, j, c] = sum(X[y.==c,j].==0)/sum(y.==c)
25    end
26  end
```

Figure 6: Modified portion of the code for the naive Bayes implementation.

2.4 Runtime of Naive Bayes for Discrete Data

Assume you have the following setup:

- The training set has n objects each with d features.
- The test set has t objects with d features.
- Each feature can have up to c discrete values (you can assume $c \leq n$).
- There are k class labels (you can assume $k \leq n$)

You can implement the training phase of a naive Bayes classifier in this setup in $O(nd)$, since you only need to do a constant amount of work for each $X[i, j]$ value. (You do not have to actually implement it in this way for the previous question, but you should think about how this could be done). What is the cost of classifying t test examples with the model?

Answer: For each sample $i \in [1, t]$, for each feature $j \in [1, d]$ (with its corresponding value predetermined (it's a test sample)), and each class $s \in [1, k]$, we need to fetch $p(\hat{X}[i, j] = < predetermined > \mid \hat{y} = s)$. Therefore, the cost is O(tdk).

3 K-Nearest Neighbours

In *citiesSmall* dataset, nearby points tend to receive the same class label because they are part of the same state. This indicates that a $k-$nearest neighbours classifier might be a better choice than a decision tree (while naive Bayes would probably work poorly on this dataset). The file *knn.jl* has implemented the training function for a $k-$nearest neighbour classifier (which is to just memorize the data) but the predict function always just predicts 1.

3.1 KNN Prediction

Fill in the *predict* function in *knn.jl* so that the model file implements the k-nearest neighbour prediction rule. You should use Euclidean distance.

Hint: although it is not necessary, you may find it useful to pre-compute all the distances (using the *distancesSquared* function in *misc.jl*) and to use the *sortperm* command.

1. Hand in the predict function.

   Answer: The implementation of the prediction function is shown in Figure 7

```
include("misc.jl") # Includes GenericModel typedef

function knn_predict(Xhat,X,y,k)
  (n,d) = size(X)
  (t,d) = size(Xhat)
  k = min(n,k) # To save you some debuggin

  # Compute the pairwise distance between all samples in the test and training set
  # The distancesSquared function computes the "squared" distance, which is OK since
  # we only care about relative distance differences
  distances = distancesSquared(Xhat, X)

  predictions = zeros(t)
  for i in 1:t
    sorted_idx = sortperm(distances[i, :])
    predictions[i] = mode(y[sorted_idx[1:k]])
  end

  return predictions
end

function knn(X,y,k)
  # Implementation of k-nearest neighbour classifier
  predict(Xhat) = knn_predict(Xhat,X,y,k)
  return GenericModel(predict)
end
```

Figure 7: KNN predict function implementation

2. Report the training and test error obtained on the *citiesSmall.mat* dataset for $k = 1$, $k = 2$, and $k = 3$. (You can use *example_knn.jl* to get started.)

   Answer:

   - k=1 → Training Error: 0.000 Test Error: 0.065
   - k=2 → Training Error: 0.035 Test Error: 0.092
   - k=3 → Training Error: 0.028 Test Error: 0.066

3. Hand in the plot generated by *plot2Dclassifier* on the *citiesSmall.mat* dataset for $k = 1$ on the training data.

   Answer: The plot is shown in Figure 8.

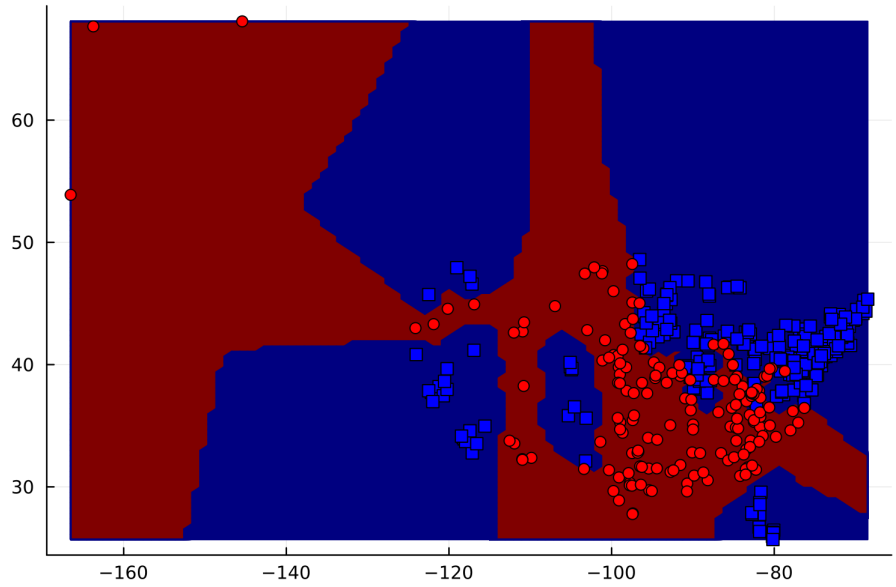


Figure 8: Plot of training set predictions of the KNN model with k=1

4. If we entered the coordinates of Vancouver into the predict function, would it be predicted to be in a blue state or a red state?

   Answer: The model predicts a label of "1" which is a blue state.

5. Why is the training error 0 for $k = 1$?

   Answer: During training error computation, the $k = 1$ neighbour is the input sample itself, having a distance of 0 (diagonal of distances matrix is all 0's). Therefore, the input sample's label is chosen as the output, which is always correct and achieves a training error of 0. This is an obvious case of overfitting (knn model is more complicated for lower k's).

6. If you didn't have an explicit test set, how would you choose k?

   Answer: We could split the dataset (either randomly or not based on context) into a training and a validation set. We could then use the training set for training the model and evaluate its performance on the validation set to find the optimal k value. For better results (and to minimize optimization bias), we could also use k-fold validation.

Hint: when writing a function, it is typically a good practice to write one step of the code at a time and check if the code matches the output you expect. You can then proceed to the next step and at each step test is if the function behaves as you expect. You can also use a set of inputs where you know what the output should be in order to help you find any bugs. These are standard programming practices: it is not the job of the TAs or the instructor to find the location of a bug in a long program you've written without verifying the individual parts on their own.

4 Random Forests

4.1 Implementation

The file *vowels.jld* contains a supervised learning dataset where we are trying to predict which of the 11 "steady-state" English vowels that a speaker is trying to pronounce.

You are provided with a a `randomTree` function in *randomTree.jl* (based on information gain). The random tree model differs from the decision tree model in two ways: it takes a bootstrap sample of the data before fitting and when fitting individual stumps it only considers $\lfloor\sqrt{d}\rfloor$ randomly-chosen features.[1] In other words, `randomTree` is the model we discussed in class that is combined to make up a random forest.

If you run *example_randomTree.jl*, it will fit both models to the dataset, and you will notice that it overfits badly.

1. If you set the *depth* parameter to *Inf*, why do the training functions terminate?

   Answer: With a depth of infinity, the random trees keep splitting the data until each leaf contains homogeneous samples that cannot be split any further. This causes the decision stump to not be able to split the branch any further, which is captured by an if statement in the *randomTree_sub* function.

2. Why does the random tree model, using infoGain and a depth of *Inf*, have a training error greater 0?

   Answer: Our random tree implementation relies on bootstraps of the main dataset for training, and it achieves a training error of 0 for that specific bootstrap. However, in the main script, we are reporting the training error on the main (non-bootstrapped) dataset, which gives us a non-zero value.

3. Create a function `randomForest` that takes in hyperparameters `depth` and `nTrees` (number of trees), and fits `nTrees` random trees each with maximum depth `depth`. For prediction, have all trees predict and then take the mode. Hand in your function. Hint: you can define an array for holding 10 *GenericModel* types using:
   `subModels = Array{GenericModel}(undef,10)`.

   Answer: The function implementation is shown in Figure 9

4. Using 50 trees, and a depth of ∞, report the training and testing error. Compare this to what we got with a single `DecisionTree` and with a single `RandomTree`. Are the results what you expected? Discuss.

   Answer: The losses are summarized in Table 1. Yes, the results are what I expected. As for test error,

	Training Error	Test Error
Random Forest	0	0.178
Decision Tree	0	0.367
Random Tree	0.170	0.485

Table 1: Table of decision tree results

   the RandomTree is not utilizing the full training set (only uses a bootstrap), and its test result is worse than that of the DecisionTree where the full training set is used. In the other hand, RandomForest achieves the best test error because it takes the mode of predictions from multiple random trees, and its result is expected to be better than the individual trees that make independent errors.
   For the training error, the randomTree implementation does not achieve a training error of 0 because of what was discussed in question 2. However, randomForest achieves a training error of 0 because it is taking the mode of many predictions from different random trees, where each tree sees a bootstrap of

[1] The notation $\lfloor x\rfloor$ means the "floor" of x, or "x rounded down".

```julia
include("misc.jl")
include("randomTree.jl")
include("decisionStump.jl")

function randomForest(X, y, depth, nTrees)
    subModels = Array{GenericModel}(undef,nTrees)

    for i in 1:nTrees
        subModels[i] = randomTree(X, y, depth)
    end

    function predict(Xhat)
        (t,d) = size(Xhat)
        yTrees = zeros(t, nTrees)
        yhat = zeros(t)

        for i in 1:nTrees
            yTrees[:, i] = subModels[i].predict(Xhat)
        end

        for i in 1:t
            yhat[i] = mode(yTrees[i, :])
        end

        return yhat

    end

    return GenericModel(predict)

end
```

Figure 9: Julia implementation of random forest.

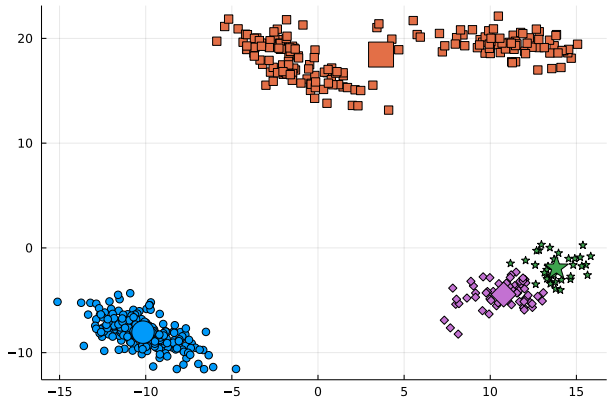
the original dataset. With many bootstraps, it is highly likely that the randomForest sees the original dataset.

5. Why does a random forest typically have a training error of 0, even though random trees typically have a training error greater than 0?

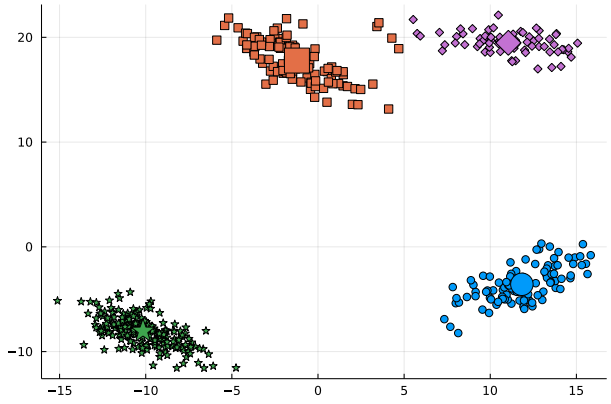
   Answer: The RandomTree is trained on a single bootstrap of the dataset, where the whole training set may not be observed. Therefore, when assessing RandomTree's performance on the complete dataset, a greater than 0 error is obtained. The RandomForest, however, sees many different bootstraps and is highly likely to observe the complete dataset when results from the random trees are accumulated. To show this, when we use a lower number of random trees for the RandomForest (5 trees), we see a non-zero training error.

5 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:



5.1 Selecting among k-means Initializations

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},

$$f(w_1, w_2, \ldots, w_k, y_1, y_2, \ldots, y_n) = \sum_{i=1}^{n} \|x_i - w_{y_i}\|_2^2 = \sum_{i=1}^{n} \sum_{j=1}^{d} (x_{ij} - w_{y_i j})^2.$$

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 10

```julia
include("misc.jl")

function kMeansError(X, y, W)
    (n,d) = size(X)

    error = 0

    for i in 1:n
        for j in 1:d
            error += (X[i, j] - W[y[i], j])^2
        end
    end

    return error
end
```

Figure 10: Implementation of the kMeansError function.

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: As shown in Figure 11, we see that while the error has a decreasing trend (until it plateaus), the rate is not the same as the drop rate of the cluster changes. Additionally, we see that running the code multiple times causes different errors to be shown, which shows the label switching phenomenon.

```
Running k-means, changes = 500 and error = 32263.029390
Running k-means, changes = 127 and error = 13549.102485
Running k-means, changes = 35 and error = 3197.955795
Running k-means, changes = 4 and error = 3071.468053
Running k-means, changes = 0 and error = 3071.468053
```

Figure 11: KMeansError for a single run of the kMeans algorithm.

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: We get the clusters shown in Figure 12, which matches the desired clusters shown in the question. The implementation to obtain the lowest-error model is shown in Figure 13.

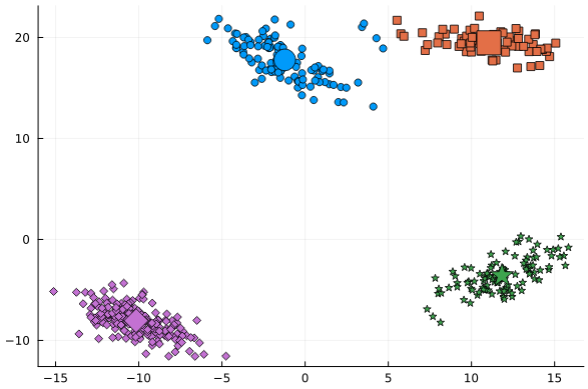


Figure 12: Cluster assignment achieved by finding optimal kMeansError.

```julia
# Load data
using JLD
include("kMeansError.jl")
X = load("clusterData.jld","X")

# K-means clustering
k = 4
include("kMeans.jl")

min_error = 9999999999
chosen_y = 0
chosen_model = 0
for i in 1:50
    model = kMeans(X,k,doPlot=false)
    y = model.predict(X)
    error = kMeansError(X, y, model.W)

    if error < min_error
        global min_error = error
        global chosen_y = y
        global chosen_model = model
        @printf("******** Error decresed to %f ********\n", error)
    end
end

include("clustering2Dplot.jl")
clustering2Dplot(X,chosen_y,chosen_model.W)
```

Figure 13: Finding the best kMeansError among 50 runs.

5.2 Selecting k in k-means

We now turn to the task of choosing the number of clusters k.

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

   Answer: If we set $k = n$ where n is the number of training samples, each sample will be a cluster of its own with an overall kMeansError of 0. Therefore, this is not a suitable metric to find k.

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

   Answer: Similar to the previous answer, a new sample's distance to an individual sample will be smaller or equal to the mean of multiple samples. Therefore, there's a high proabability that $k = n$ would be chosen again using this metric.

3. Hand in a plot of the minimum error found across 50 random initializations, as you vary k from 1 to 10.

   Answer: The plot is shown in Figure 14. The implementation to obtain this plot is shown in Figure 15.

4. The *elbow method* for choosing k consists of looking at the above plot and visually trying to choose

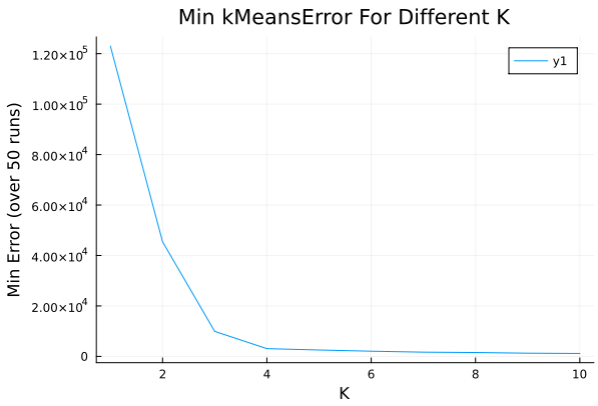


Figure 14: Minimum Error over 50 iterations of the KMeans algorithm with different K.

```julia
# Load data
using JLD
include("kMeansError.jl")
X = load("clusterData.jld","X")

# K-means clustering
include("kMeans.jl")

min_errors = zeros(10)
for k in 1:10
    min_error = 9999999999
    for i in 1:50
        model = kMeans(X,k,doPlot=false)
        y = model.predict(X)
        error = kMeansError(X, y, model.W)

        if error < min_error
            global min_errors[k] = error
            min_error = error
        end
    end
end

using Plots
plot(1:10, min_errors, title="Min kMeansError For Different K")
xlabel!("K")
ylabel!("Min Error (over 50 runs)")
```

Figure 15: Code to obtain the plot in Figure 14.

the k that makes the sharpest "elbow" (the biggest change in slope). What values of k might be reasonable according to this method? Note: there is not a single correct answer here; it is somewhat open to interpretation and there is a range of reasonable answers.

Answer: Based on the slope changes in the plot, $k = 3$ and $k = 4$ seem like reasonable choices.

5.3 k-Medians

The data in *clusterData2.mat* is the exact same as the above data, except it has 4 outliers that are very far away from the data.

1. Using the *clustering2Dplot* function, output the clustering obtained by running k-means 50 times (with $k = 4$) on *clusterData2.mat* and taking the one with the lowest error. Are you satisfied with the result?

   Answer: The plot is shown in Figure 16. I am not satisfied with the results as some outliers are associated with clusters that are far away from them. Additionally, since the model is forced to find 4 clusters, it is associating a single outlier with a cluster of its own.

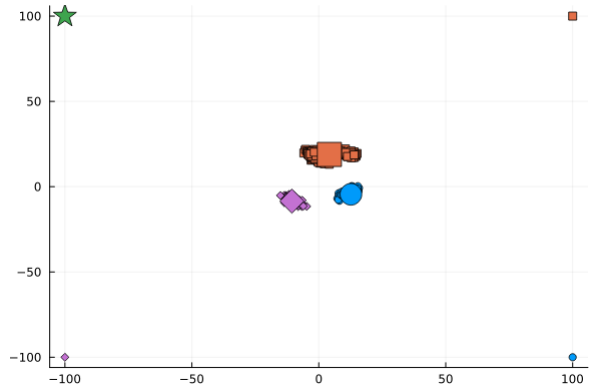


Figure 16: Clusters obtained by running KMeans 50 times.

2. Hand in the elbow plot for this data. What values of k might be chosen by the elbow method for this dataset?

   Answer: The elbow plot is shown in Figure 17, where we see that $k = 7$ and $k = 8$ seem like reasonable choices for k.

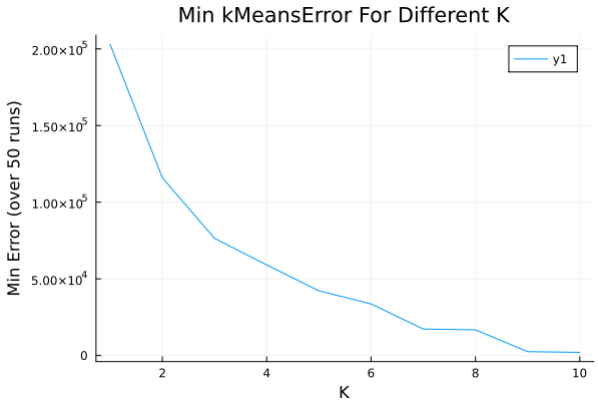


Figure 17: Elbow plot for clusterData2 dataset.

3. Instead of the squared distances between the examples x_i and their cluster centers, consider measuring

the distance to the cluster centers in the L1-norm,

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

Hand in the elbow plot for k-means when we measure the error of the final model with the L1-norm. What value of k would be chosen by the elbow method?

Answer: The elbow plot is shown in Figure 18. The optimal K seems to be 3 or 4 in this case.

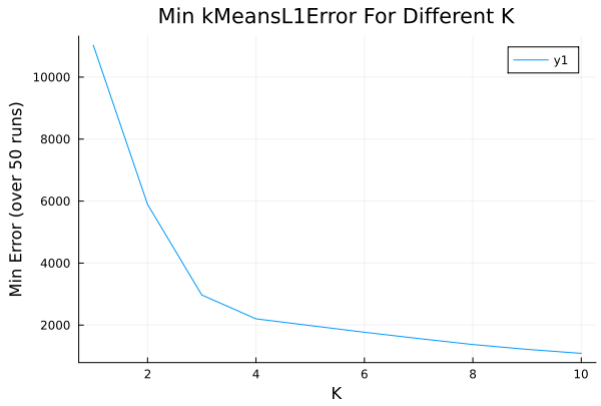


Figure 18: Elbow plot for clusterData2 dataset with K medians.

4. The k-means algorithm tries to minimize the squared error and not the L1-norm error, so in the last question there is a mis-match between the what the learning algorithm tries to minimize and how we measure the final error. We can try to directly minimize the L1-norm with the *k-medians* algorithm, which assigns examples to the nearest w_c in the L1-norm and to 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 median of the points along each dimension). Implement the k-medians algorithm, and hand in your code and the the plot obtained by minimizing the L1-norm error across 50 random initializations of k-medians with $k = 4$.

Answer: The clusters are shown in Figure 19, and the implementation is shown in Figure 20.

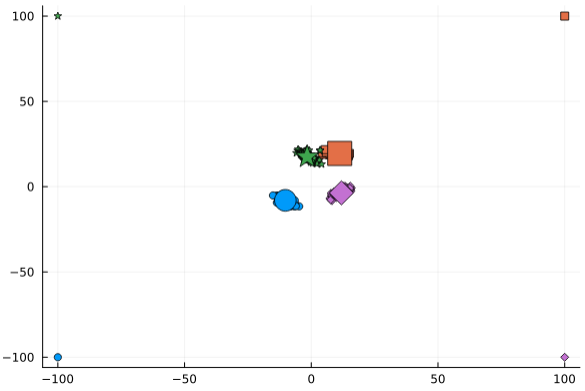


Figure 19: Clusters obtained with K-medians.

```
# Choos random points to initialize medians
W = zeros(k,d)
perm = randperm(n)
for c = 1:k
    W[c,:] = X[perm[c],:]
end

# Initialize cluster assignment vector
y = zeros(Int64, n)
changes = n
while changes != 0

    # Compute L1 distance between each point and each mean
    D = pairwise(Cityblock(), X, W, dims=1)

    # Degenerate clusters will distance NaN, change to Inf
    # (since Julia thinks NaN is smaller than all other numbers)
    D[findall(isnan.(D))] .= Inf

    # Assign each data point to closest median (track number of changes labels)
    changes = 0
    for i in 1:n
        (~,y_new) = findmin(D[i,:])
        changes += (y_new != y[i])
        y[i] = y_new
    end

    # Optionally visualize the algorithm steps
    if doPlot && d == 2
        clustering2Dplot(X,y,W)
        sleep(.1)
    end

    # Find median of each cluster
    for c in 1:k
        W[c,:] = median(X[y.==c,:],dims=1)
    end

    # Optionally visualize the algorithm steps
    if doPlot && d == 2
        clustering2Dplot(X,y,W)
        sleep(.1)
    end

    error = kMeansL1Error(X, y, W)

    @printf("Running k-medians, changes = %d and error = %f\n",changes, error)
end

function predict(Xhat)
    (t,d) = size(Xhat)

    D = pairwise(Cityblock(), Xhat, W, dims=1)
    D[findall(isnan.(D))] .= Inf

    yhat = zeros(Int64,t)
    for i in 1:t
        (~,yhat[i]) = findmin(D[i,:])
    end
    return yhat
end

return PartitionModel(predict,y,W)
end
```

Figure 20: K-medians implementation.

6 Very-Short Answer Questions

Write a short one or two sentence answer to each of the questions below. Make sure your answer is clear and concise.

1. What is a feature transformation that you might do to address a "coupon collecting" problem in your data?

   Answer: We could use augmentations such as zooming/cropping/rotation to artificially increase the number of input samples and cover the data space. Performing dimension reduction (such as PCA) before feeding the data into the model could help with this problem by reducing the number of feature combinations.

2. What is one reason we would want to look at scatterplots of the data before doing supervised learning?

   Answer: Looking at the scatter plot allows us to determine the complexity of the model we need. For example, if data is separated in a linear fashion, a simple linear regressor or classifier may suffice. This helps us by avoiding models that overfit the data. Additionally, this allows us to identify outlier

samples in the dataset.

3. When we fit decision stumps, why do we only consider $>$ (for example) and not consider $<$ or $\geq$?

   Answer: We don't consider them because it doesn't make a significant difference to choose any of them over the other. The flow of internal splits in a tree might change, but the overall performance is very similar.

4. What is a reason that the data may not be IID in the email spam filtering example from lecture?

   Answer: Certain words in an email usually go together. For example, in an email that contains the word "car", it is more likely to see the word "insurance" as opposed to the word "toilet." Therefore, the probability of a word may not be conditionally independent of other words given the label.

5. What is the difference between a validation set and a test set?

   Answer: A validation set is a set used to find the optimal hyperparameters for a model and may be used multiple times for this purpose (although, one have to be careful about optimization bias). Whereas, a test set is to be used once to evaluate a model's performance on unseen data.

6. Why can't we (typically) use the training error to select a hyper-parameter?

   Answer: We know that over-complex models achieve a training error of 0 by overfitting the training dataset. Therefore, using the training dataset for hyperparameter tuning (which affects model complexity) would encourage the selection of models that overfit the training dataset.

7. If you can fit one model in 10 sec., how long (in days) does it take to find the best among a set of 16 hyperparameter values using leave-one-out cross-validation on a dataset containing $n = 4320$ examples?

   Answer: time $= \frac{4320 \times 16 \times 10}{86400} = 8$days

8. Naïve Bayes makes the assumption that all features are conditionally independent given the class label. Why is this assumption necessary and what would happen without it?

   Answer: Without this assumption, our formula takes the form $p(y \mid x) \propto p(x, y)$ where $p(x, y)$ must be calculated using the chain rule that considers the dependency between features. With such an approach, a few issues arise. For one, depending on the size of our training dataset, we may not be able to properly capture the conditional probabilities, which may lead to worse performance. Another major reason is that at higher dimensions, fitting the model becomes expensive as all conditional probabilities must be captured (2^d). Additionally, we may need a graph probability model to properly capture the dependence and relationship (through link function) between the features.

9. Why is KNN considered a non-parametric method and what are two undesirable consequences of KNNs non-parametric design?

   Answer: For KNN, we need to save all training samples in order to find the k nearest neighbours of a test sample. Therefore, KNNs are non-parameteric since memory complexity depends on training dataset size. One complexity is that since memory complexity is O(nd), with a large number of training samples of high dimensionality, model requires large amounts of memory. Another complexity is that test time also increases with the number of samples as the test sample's distance with all training samples must be computed.

10. For any parametric model, how does increasing number of training examples n affect the two parts of the fundamental trade-off.

    Answer: In general, increasing the number of training examples reduces E_{approx} as the model learns better inductive bias, and E_{train} could increase as the model may need more complexity to find all the patterns in this larger dataset. All of these, however, depend on how complex the original model is.

11. Suppose we want to classify whether segments of raw audio represent words or not. What is an easy way to make our classifier invariant to small translations of the raw audio?

    Answer: We could randomly add varying amounts of noise to the audio signal and expect the model to be able to detect the segments corresponding to words. Additionally, we could phase shift the signal to make the model invariable to temporal translations.

12. Both supervised learning and clustering models take in an input x_i and produce a label y_i. What is the key difference?

    Answer: Presence of training labels is the main difference. For supervised learning, we have a training set where each sample is associated with a label. For clustering, however, we only have the samples without any associated labels, and we find which samples correspond to the same group. Additionally, a label given to a sample during clustering can only be used to say which other samples are in the same group and does not necessarily have a semantic meaning (label switching).

13. In k-means clustering the clusters are guaranteed to be convex regions. Are the areas that are given the same label by KNN also convex?

    Answer: No not necessarily. For example, it is highly likely that the regions are non convex with $k = 1$ with even disjoint areas having the same label.

CPSC 340 Assignment 3 (due Monday October 17 at 11:55pm)

1 More Unsupervised Learning

1.1 Vector Quantization

Discovering object groups is one motivation for clustering. Another motivation is *vector quantization*, where we find a prototype point for each cluster and replace points in the cluster by their prototype. If our inputs are images, we could use vector quantization on the set of RGB pixel values as a simple image compression algorithm.

Your task is to implement this simple image compression algorithm by writing a `quantizeImage` and a `deQuantizeImage` function. The `quantizeImage` function should take the name of an image file (like "dog.png" for the provided image) and a number b as input. It should use the pixels in the image as examples and the 3 colour channels as features, and run k-means clustering on this data with 2^b clusters. The code should store the cluster means and return four arguments: the cluster assignments y, the means W, the number of rows in the image $nRows$, and the number of columns $nCols$. The `deQuantizeImage` function should take these four arguments and return a version of the image (the same size as the original) where each pixel's original colour is replaced with the nearest prototype colour.

To understand why this is compression, consider the original image space. Say the image can take on the values $0, 1, \ldots, 254, 255$ in each colour channel. Since $2^8 = 256$ this means we need 8 bits to represent each colour channel, for a total of 24 bits per pixel. Using our method, we are restricting each pixel to only take on one of 2^b colour values. In other words, we are compressing each pixel from a 24-bit colour representation to a b-bit colour representation by picking the 2^b prototype colours that are "most representative" given the content of the image. So, for example, if $b = 6$ then we have 4x compression.

Note: the script *example_image.jl* shows how to read an image file using *Images* package, how to display an image using the *Plots* package, and how to convert images to/from the feature representation of their pixels.

1. Hand in your *quantizeImage* and *deQuantizeImage* functions.

   Answer: The implementation for all functions is shown in Figure 1

```julia
using Images, Plots
include("kMeans.jl")


function to_features(I)

    (nRows,nCols) = size(I)

    # Convert to features
    R = permutedims(channelview(I),[2,3,1])
    X = reshape(float64.(R),(nRows*nCols,3))

    return X
end


function to_image(X, nRows, nCols)

    # Convert from features back to image
    R = reshape(X,(nRows,nCols,3))
    I = colorview(RGB,permutedims(R,[3,1,2]))

    return I
end


function quantizeImage(img_name, b)

    # Load the image
    I = load(img_name)
    (nRows,nCols) = size(I)

    # Transform the image into features
    X = to_features(I)

    # Fit the kMeans model
    kmeans_model = kMeans(X, 2^b)

    return kmeans_model.y, kmeans_model.W, nRows, nCols
end


function deQuantizeImage(y, W, nRows, nCols)

    # Reconstructed features
    X = W[y, :]

    # From features to image
    I = to_image(X, nRows, nCols)

    return I

end
```

Figure 1: Implementation of functions required for image compression.

2. Show the image obtained if you encode the colours using 1, 2, 4, and 6 bits per pixel (instead of the original 24-bits).

   Answer: The images are shown in Figure 2.

(a) b=1 (b) b=2

(c) b=4 (d) b=6

Figure 2: Resulting dog images after compression.

3. Save the image with 6 bits per pixel to a .PNG file with the *save* function. By going from 24-bits (8-bits for each of the three colour channels) to 6-bits to store the colour of each pixel we would expect the PNG file to be 25% the size of the original image. Explain why you think the actual PNG file is larger or smaller than this compression value.

   Answer: The original file's size is 225KB, and the compressed file's size is 126KB, which is 56% of the original size. Therefore, we are achieving less compression than we anticipated. My reasoning for seeing this is that the *save* function (for png) does not use the same clustering algorithm for compression as we did and is doing compression in a manner that information is not lost (lossless compression). While this approach is not achieving the same compression rate as our approach, it is still compressing the image to an extent.

1.2 Density-Based Clustering

If you run the function *example_dbCluster*, it will apply the basic density-based clustering algorithm to the dataset from the previous part. The final output should look like this:
(The right plot is zoomed in to show the non-outlier part of the data.) Even though we know that each object was generated from one of four clusters (and we have 4 outliers), the algorithm finds 6 clusters and does not assign some of the original non-outlier objects to any cluster (points not assigned to any cluster and displayed as small black circles). However, the clusters will change if we change the parameters of the algorithm. Find and report values for the two parameters (*radius* and *minPts*) such that the density-based clustering method finds:

1. The 4 "true" clusters.

   Answer:

   - radius: 2
   - minPts: 2

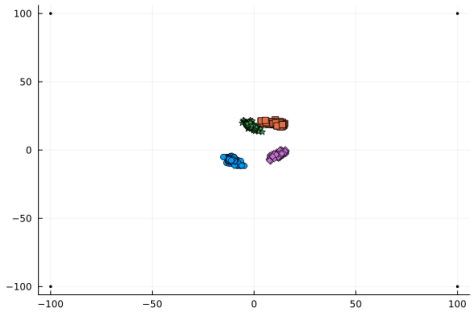


Figure 3: 4 clusters

2. 3 clusters (merging the top two, which also seems like a reasonable interpretation).

   Answer:

   - radius: 4
   - minPts: 2

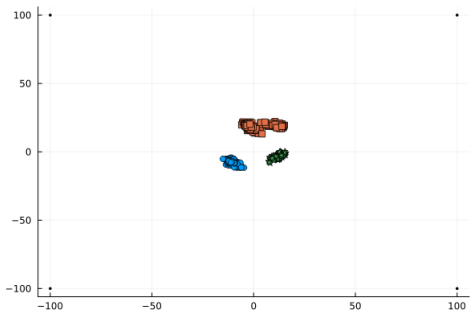


Figure 4: 3 clusters

3. 2 clusters.

   Answer:

- radius: 15
- minPts: 2

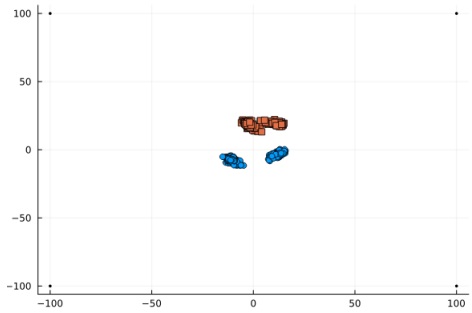


Figure 5: 2 clusters

4. 1 cluster (consisting of the non-outlier points).

   Answer:

   - radius: 20
   - minPts: 2

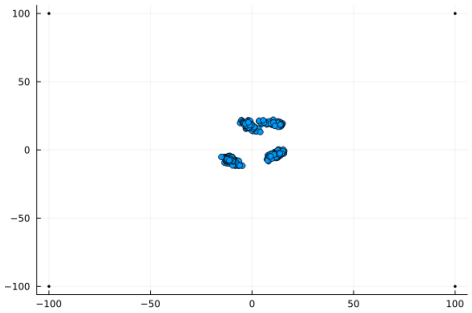


Figure 6: 1 cluster

2 Matrix Notation and Linear Regression

2.1 Converting to Matrix/Vector/Norm Notation

Using our standard supervised learning notation (X, y, w) express the following functions in terms of vectors, matrices, and norms (there should be no summations or maximums).

1. $\sum_{i=1}^{n} |w^T x_i - y_i| + \lambda \sum_{j=1}^{d} |w_j|$.

   Answer: $\|Xw - y\|_1 + \lambda \|w\|_1$

2. $\sum_{i=1}^{n} v_i (w^T x_i - y_i)^2 + \sum_{j=1}^{d} \lambda_j w_j^2$.

   Answer: $(Xw - y)^T V (Xw - y) + w^T \Lambda w$

3. $\left(\max_{i \in \{1,2,\ldots,n\}} |w^T x_i - y_i|\right)^2 + \frac{1}{2} \sum_{j=1}^{d} \lambda_j |w_j|$.

   Answer: $\|Xw - y\|_\infty^2 + \frac{1}{2} \|\Lambda w\|_1$

You can use V to denote a diagonal matrix that has the (non-negative) "weights" v_i along the diagonal. The value λ (the "regularization parameter") is a non-negative scalar. You can Λ as a diagonal matrix that has the (non-negative) λ_j values along the diagonal.

2.2 Minimizing Quadratic Functions as Linear Systems

Write finding a minimizer w of the functions below as a system of linear equations (using vector/matrix notation and simplifying as much as possible). Note that all the functions below are convex so finding a w with $\nabla f(w) = 0$ is sufficient to minimize the functions (but show your work in getting to this point).

1. $f(w) = \frac{1}{2}\|w - u\|^2$ (projection of u onto real space).

   Answer: Using the method from course's extra notes:

   - Convert to summation form: $f(w) = \frac{1}{2}\sum_{j=1}^{d}(w_j - u_j)^2$.
   - Partial derivative for element k: $\frac{d}{dw_k}[\frac{1}{2}\sum_{j=1}^{d}(w_j - u_j)^2] = w_k - u_k$
   - Assembling partial derivative: $\nabla f(w) = \begin{bmatrix} \frac{d}{dw_1} \\ \frac{d}{dw_2} \\ \vdots \\ \frac{d}{dw_k} \end{bmatrix} = \begin{bmatrix} w_1 - u_1 \\ w_2 - u_2 \\ \vdots \\ w_k - u_k \end{bmatrix} = 0 \rightarrow w = u$

2. $f(w) = \frac{1}{2}\sum_{i=1}^{n} v_i(w^T x_i - y_i)^2 + \lambda w^T u$ (weighted and tilted least squares).

   Answer:

$$\begin{aligned} f(w) &= \frac{1}{2}(Xw - y)^T V(Xw - y) + \lambda w^T u \\ &= \frac{1}{2}(Xw - y)^T (VXw - Vy) + \lambda w^T u \\ &= \frac{1}{2}((VXw - Vy)^T (Xw - y))^T + \lambda w^T u \\ &= \frac{1}{2}((w^T X^T V - y^T V)(Xw - y))^T + \lambda w^T u \\ &= \frac{1}{2}(w^T X^T V - y^T V)(Xw - y) + \lambda w^T u \\ &= \frac{1}{2}(w^T X^T V Xw - w^T X^T V y - y^T V Xw + y^T V y) + \lambda w^T u \\ &= \frac{1}{2} w^T X^T V Xw - w^T X^T V y + \frac{1}{2} y^T V y + \lambda w^T u \end{aligned}$$

   Therefore, as per lecture notes, $\nabla f(w) = X^T V Xw - X^T V y + \lambda u = 0 \rightarrow X^T V Xw = X^T V y - \lambda u$

3. $f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w - w^0\|^2$ (least squares shrunk towards non-zero w^0).

   Answer:

$$f(w) = \frac{1}{2} w^T X^T Xw - w^T X^T y + \frac{1}{2} y^T y + \frac{\lambda}{2}\|w - w^0\|^2$$

   Now, using the results from the lecture and our answer to part Q2.2.1, we have:

$$\begin{aligned} \nabla f(w) &= X^T Xw - X^T y + \lambda(w - w^0) = X^T Xw - X^T y + \lambda w - \lambda w^0 \\ &= (X^T X + \lambda I)w - X^T y - \lambda w^0 = 0 \rightarrow (X^T X + \lambda I)w = X^T y + \lambda w^0 \end{aligned}$$

Above we assume that u and w^0 are d by 1 vectors, that v is a n by 1 vector. You can use V as a diagonal matrix containing the v_i values along the diagonal.

Hint: Once you convert to vector/matrix notation, you can use the results from class to quickly compute these quantities term-wise. As a sanity check for your derivation, make sure that your results have the right dimensions. As a sanity check, make that the dimensions match for all quantities/operations. In order to

make the dimensions match you may need to introduce an identity matrix. For example, $X^T Xw + \lambda w$ can be re-written as $(X^T X + \lambda I)w$.

2.3 Convex Functions

Recall that convex loss functions are typically easier to minimize than non-convex functions, so it's important to be able to identify whether a function is convex.

Show that the following functions are convex:

1. $f(w) = \frac{1}{2}w^2 + w^{-1}$ with $w > 0$.

   Answer: The function is convex because:

   - $\frac{d^2}{dw}\frac{1}{2}w^2 = 1 \geq 0$, which means the function is convex
   - $\frac{d^2}{dw}\frac{1}{2}w^{-1} = 2w^{-3} \geq 0$ because $w > 0$, which means the function is convex
   - Addition of two convex function is convex

2. $f(w) = \max_i w_i$ with $w \in \mathbb{R}^n$ (maximum).

   Answer: The function is convex because:

   - $\max_i w_i = max\{w_1, w_2, ..., w_n\}$
   - Second derivative of each of these elements is 0, therefore, they are convex. In other words, we get a hessian of all 0's.
   - Max of convex functions is convex

3. $f(y) = \max(0, 1 - t \cdot y)$ with $y \in \mathbb{R}$ and $t \in \{-1, +1\}$ (hinge loss).

   Answer: The function is convex because:

   - $\frac{d^2}{dy}0 = 0$, which is convex
   - with $t = -1$, $\frac{d^2}{dy}1 + y = 0$, which is convex
   - with $t = 1$, $\frac{d^2}{dy}1 - y = 0$, which is convex
   - Max of convex functions is convex

4. $f(w) = \|Xw - y\|^2 + \lambda\|w\|_1$ with $w \in \mathbb{R}^d, \lambda \geq 0$ (L1-regularized least squares).

   Answer: The function is convex because:

   - $Xw - y$ is linear
   - $\|Xw - y\|^2$ is convex because it's a convex function (power 2 of norm) composed with a linear function.
   - $\|w\|_1$ is a norm and is, therefore, convex
   - $\lambda\|w\|_1$ is convex because it's a convex function multiplied by a non-negative scalar
   - Sum of convex functions is convex

5. $f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$ with $w \in \mathbb{R}^d$ (logistic regression).

   Answer: The function is convex because:

   - $\frac{d}{dz}log(1 + exp(z)) = \frac{exp(z)}{1+exp(z)} = \frac{1}{1+exp(-z)}$
   - $\frac{d}{dz}\frac{1}{1+exp(-z)} = \frac{1}{(1+exp(-z))^2} * exp(-z) \geq 0$ because $exp(-z) \geq 0$, and $\frac{1}{(1+exp(-z))^2} \geq 0$. Therefore, $log(1 + exp(z))$ is convex.
   - $-y_i w^T x_i$ is a linear function

- $log(1 + exp(-y_i w^T x_i))$ is convex because it's a convex function composed with a linear function.
- Overall function is convex because it's the sum of convex functions

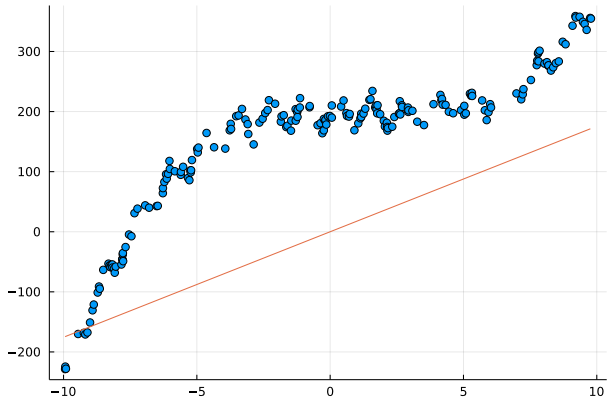
Hint for Part 5: this function may seem non-convex since it contains $\log(z)$ and log is concave, but there is a flaw in that reasoning: for example $\log(\exp(z)) = z$ is convex despite containing a log. To show convexity, it may be helpful to show that $\log(1+\exp(z))$ is convex, which can be done by computing the second derivative. It may simplify matters to note that $\frac{\exp(z)}{1+\exp(z)} = \frac{1}{1+\exp(-z)}$.

3 Linear and Nonlinear Regression

If you run the script *example_nonLinear*, it will:

1. Load a one-dimensional regression dataset.
2. Fit a least-squares linear regression model.
3. Report the training error.
4. Report the test error (on a dataset not used for training).
5. Draw a figure showing the training data and what the linear model looks like.

Unfortunately, this is an awful model of the data. The average squared training error on the data set is over 28000 (as is the test error), and the figure produced by the demo confirms that the predictions are usually nowhere near the training data:



3.1 Linear Regresion with Bias Variable

The y-intercept of this data is clearly not zero (it looks like it's closer to 200), so we should expect to improve performance by adding a *bias* variable, so that our model is

$$y_i = w^T x_i + w_0.$$

instead of

$$y_i = w^T x_i.$$

Write a new function, *leastSquaresBias*, that has the same input/model/predict format as the *leastSquares* function, but that adds a *bias* variable w_0. Hand in your new function, the updated plot, and the updated training/test error.

Hint: recall that adding a bias w_0 is equivalent to adding a column of ones to the matrix X. Don't forget that you need to do the same transformation in the *predict* function.

Answer:

- Training loss: 3551.346
- Test loss: 3393.869

The impelementation is shown in Figure 7, and the updated plot is shown in Figure 8.

```julia
function leastSquaresBias(X,y)

    (n, d) = size(X)

    # Add a 1's column to add bias
    Z = hcat(ones(n, 1), X)

    # Find regression weights minimizing squared error
    v = (Z'Z)\(Z'y)

    # Make linear prediction function
    function predict(Xhat)
        nhat = size(Xhat)[1]
        Xhat = hcat(ones(nhat, 1), Xhat)
        return Xhat*v
    end

    # Return model
    return GenericModel(predict)
end
```

Figure 7: Implementation of linear regression with bias term.

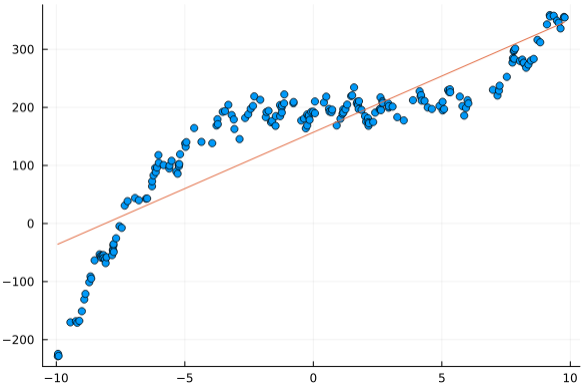


Figure 8: Updated plot for model with the bias term.

3.2 Linear Regression with Polynomial Basis

Adding a bias variable improves the prediction substantially, but the model is still problematic because the target seems to be a *non-linear* function of the input. Write a new function, *leastSquaresBasis(x,y,p)*, that takes a data vector x (i.e., assuming we only have one feature) and the polynomial order p. The function should perform a least squares fit based on a matrix Z where each of its rows contains the values $(x_i)^j$ for $j = 0$ up to p. E.g., *leastSquaresBasis(x,y,3)* should form the matrix

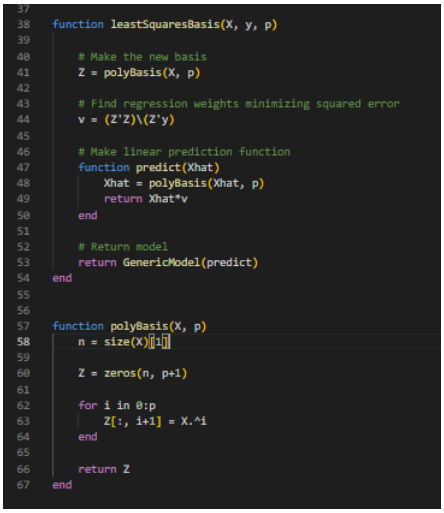
$$Z = \begin{bmatrix} 1 & x_1 & (x_1)^2 & (x_1)^3 \\ 1 & x_2 & (x_2)^2 & (x_2)^3 \\ \vdots & & & \\ 1 & x_n & (x_n)^2 & (x_N)^3 \end{bmatrix},$$

and fit a least squares model based on it. Hand in the new function, and report the training and test error for $p = 0$ through $p = 10$. Explain the effect of p on the training error and on the test error.

Note: for this question you should assume that you only have one feature ($d = 1$). We will discuss polynomial bases with more input features later in the course.

Hints: To keep the code simple and reduce the chance of having errors, you may want to write a new function *polyBasis* that you can use for transforming both the training and testing data.

Answer: The impelementation is shown in Figure 9, and the losses are summarized in Figure 10. We can also see the training and test error trends in Figure 11. We see that increasing p up to a value of 3 causes



```
function leastSquaresBasis(X, y, p)

    # Make the new basis
    Z = polyBasis(X, p)

    # Find regression weights minimizing squared error
    v = (Z'Z)\(Z'y)

    # Make linear prediction function
    function predict(Xhat)
        Xhat = polyBasis(Xhat, p)
        return Xhat*v
    end

    # Return model
    return GenericModel(predict)
end


function polyBasis(X, p)
    n = size(X)[1]

    Z = zeros(n, p+1)

    for i in 0:p
        Z[:, i+1] = X.^i
    end

    return Z
end
```

Figure 9: Implementation for the polynomial basis regression

```
P=0 ---- Squared train Error with least squares: 15480.520
P=0 ---- Squared test Error with least squares: 14390.763

P=1 ---- Squared train Error with least squares: 3551.346
P=1 ---- Squared test Error with least squares: 3393.869

P=2 ---- Squared train Error with least squares: 2167.992
P=2 ---- Squared test Error with least squares: 2480.725

P=3 ---- Squared train Error with least squares: 252.046
P=3 ---- Squared test Error with least squares: 242.805

P=4 ---- Squared train Error with least squares: 251.462
P=4 ---- Squared test Error with least squares: 242.126

P=5 ---- Squared train Error with least squares: 251.143
P=5 ---- Squared test Error with least squares: 239.545

P=6 ---- Squared train Error with least squares: 248.583
P=6 ---- Squared test Error with least squares: 246.005

P=7 ---- Squared train Error with least squares: 247.011
P=7 ---- Squared test Error with least squares: 242.888

P=8 ---- Squared train Error with least squares: 241.306
P=8 ---- Squared test Error with least squares: 245.966

P=9 ---- Squared train Error with least squares: 235.762
P=9 ---- Squared test Error with least squares: 259.296

P=10 ---- Squared train Error with least squares: 235.074
P=10 ---- Squared test Error with least squares: 256.300
```

Figure 10: Losses obtained from p=0 to p=10

a rapid decrease in both error terms. However, above 3, we don't see much performance gain. This shows that a polynomial of degree 3 properly captures the data trend in both the training and test sets. We also see some overfitting happen after p=9.

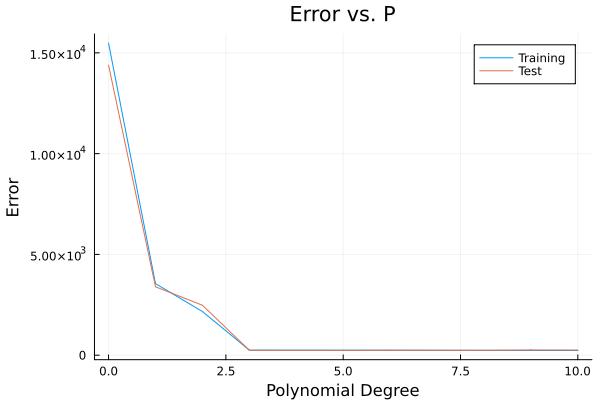


Figure 11: Plot showing training and test losses for different p values.

3.3 Gaussian RBFs

A popular alternative to using polynomials is to use Gaussian radial basis functions, with one basis function centered on each training example. For a generic feature $\tilde{x}_i$, the transformed feature vector with this choise is given by

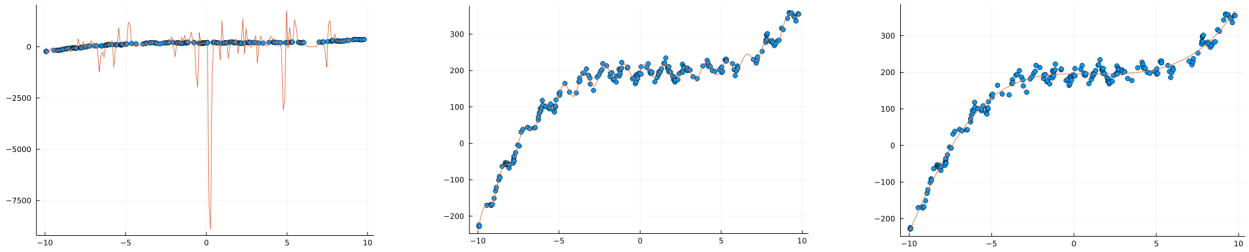
$$\tilde{z}_i = \Bigg[\underbrace{\exp\left(-\frac{(\tilde{x}_i - x_1)^2}{2\sigma^2}\right)}_{\text{feature 1}} \quad \underbrace{\exp\left(-\frac{(\tilde{x}_i - x_2)^2}{2\sigma^2}\right)}_{\text{feature 2}} \quad \cdots \quad \underbrace{\exp\left(-\frac{(\tilde{x}_i - x_n)^2}{2\sigma^2}\right)}_{\text{feature } n}\Bigg],$$

where σ^2 is a hyper-parameter. With n training examples, this generates a set of n features based on a transformation of the distance between the example and each training example. Hand in a function implementing least squares under this basis, and the plots obtained by $\sigma = 10$, $\sigma = 1$, and $\sigma = 0.1$. How does the value of σ^2 affect the fundamental trade-off?

Answer: The implementation is shown in Figure 12, and the plots for different sigma values are shown in Figure 13. We see that with smaller σ values (e.g. $\sigma = 0.1$), the model becomes more complicated and test error increases as the training error decreases (overfitting). With larger σ values (e.g. $\sigma = 10$), the model is less complex and training error increases (underfitting). A value of $\sigma = 1$ seems to achieve a good balance between training and test error.

```julia
function leastSquaresRBF(X, y, sigma)

    # Make the new basis
    Z = gausRBFBasis(X, X, sigma)

    # Find regression weights minimizing squared error
    v = (Z'Z)\(Z'y)

    # Make linear prediction function
    function predict(Xhat)
        Xhat = gausRBFBasis(Xhat, X, sigma)
        return Xhat*v
    end

    # Return model
    return GenericModel(predict)
end


function gausRBFBasis(X1, X2, sigma)

    # Create the matrix containing L2 distance squared
    D = distancesSquared(X1, X2);

    # Compute the basis
    Z = exp.(-1 .* D / (2*sigma^2));

    return Z
end
```

Figure 12: Implementation of regression with RBF basis.



(a) sigma=0.1 (b) sigma=1 (c) sigma=10

Figure 13: Plots obtained by different sigma values.

4 Robust Regression and Gradient Descent

The script *example_outliers* loads a one-dimensional regression dataset that has a non-trivial number of 'outlier' data points. These points do not fit the general trend of the rest of the data, and pull the least squares model away from the main downward trend that most data points exhibit:

4.1 Weighted Least Squares in One Dimension

One of the most common variations on least squares is *weighted* least squares. In this formulation, we have a weight v_i for every training example. To fit the model, we minimize the weighted squared error,

$$f(w) = \frac{1}{2}\sum_{i=1}^{n} v_i(w^T x_i - y_i)^2.$$

In this formulation, the model focuses on making the error small for examples i where v_i is high. Similarly, if v_i is low then the model allows a larger error.

Write a model function, *weightedLeastSquares(X,y,v)*, that implements this model (note that this can be solved as a linear system). Apply this model to the data containing outliers, setting $v_i = 1$ for the first 400 data points and $v_i = 0.1$ for the last 100 data points (which are the outliers). Hand in your function and the updated plot.

Answer: The implementation is provided in Figure 14 where we use our results from Q2.2.2 to find the solution. The updated plot is shown in Figure 15.

```
101  function weightedLeastSquares(X,y,v)
102
103      # Diagnolize v
104      V = Diagonal(v)
105
106      # Find regression weights minimizing squared error
107      w = (X'*V*X) \ (X'*V*y)
108
109      # Make linear prediction function
110      predict(Xhat) = Xhat*w
111
112      # Return model
113      return GenericModel(predict)
114  end
```

Figure 14: Implementation of weighted least squares.

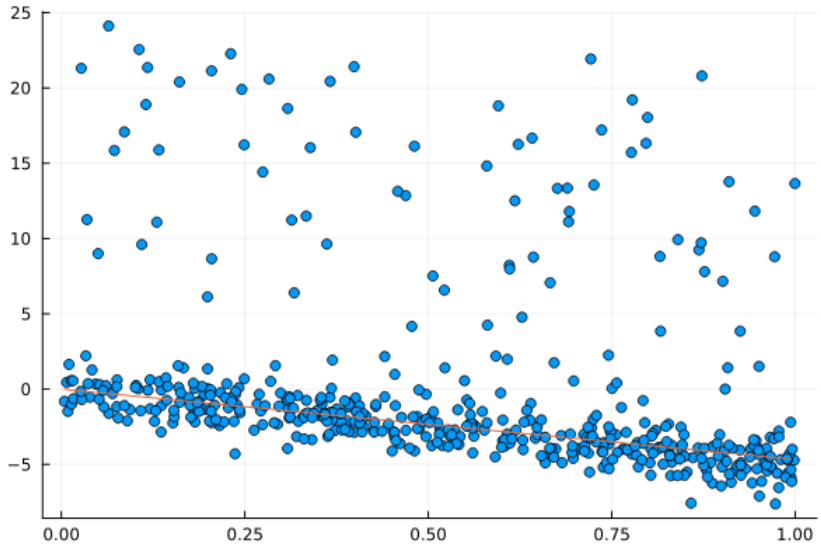


Figure 15: Results plot for the weighted least squares model.

4.2 Smooth Approximation to the L1-Norm

Unfortunately, we typically do not know the identities of the outliers. In situations where we suspect that there are outliers, but we do not know which examples are outliers, it makes sense to use a loss function that is more robust to outliers. In class, we discussed using the Huber loss,

$$f(w) = \sum_{i=1}^{n} h(w^T x_i - y_i),$$

where

$$h(r_i) = \begin{cases} \frac{1}{2} r_i^2 & \text{for } |r_i| \leq \epsilon \\ \epsilon(|r_i| - \frac{1}{2}\epsilon) & \text{otherwise} \end{cases}.$$

This is less sensitive to outliers than least squares, although it can no longer be minimized by solving a linear system. Derive the gradient ∇f of this function with respect to w. You should show your work but you do not have to express the final result in matrix notation. Hint: you can start by computing the derivative of h with respect to r_i and then get the gradient using the chain rule. You can use $\text{sgn}(r_i)$ as a function that returns 1 if r_i is positive and -1 if it is negative.

Answer: Let's find the gradient with respect to r_i first:

$$\frac{d}{d_{r_i}} \frac{1}{2} r_i^2 = r_i$$
$$\frac{d}{d_{r_i}} \epsilon(|r_i| - \frac{1}{2}\epsilon) = \text{sgn}(r_i)\epsilon$$

Using the chain rule:

- For $|r_i| \leq \epsilon$: $\frac{d}{d_{w_j}} h(w^T x_i - y_i) = (w^T x_i - y_i) x_{ij}$
- For other r_i values: $\frac{d}{d_{w_j}} h(w^T x_i - y_i) = \text{sgn}(w^T x_i - y_i)\epsilon x_{ij}$

Therefore:

$$\frac{d}{dw_j} f(w) = \sum_{i=1}^{n} \hat{h}_{ij}$$

where

$$\hat{h}_{ij} = \begin{cases} (w^T x_i - y_i) x_{ij} & \text{if } |r_i| \leq \epsilon \\ \text{sgn}(w^T x_i - y_i)\epsilon x_{ij} & \text{otherwise} \end{cases}.$$

The gradient is just the concatenation of the above for $j \in [1, d]$.

4.3 Robust Regression

The function *example_gradient* is the same as *example_outlier*, except that it fits the least squares model using a *gradient descent* method. You'll see that it produces the same fit as we obtained using the normal equations.

The typical input to a gradient method is a function that, given w, returns $f(w)$ and $\nabla f(w)$. See *funObj* in the *leastSquaresGradient* function for an example. Note that *leastSquaresGradient* also has a numerical check that the gradient code is approximately correct, since implementing gradients is often error-prone.[1]

An advantage of gradient-based strategies is that they are able to solve problems that do not have closed-form solutions, such as the formulation from the previous section. The function *robustRegression* has most of the implementation of a gradient-based strategy for fitting the Huber regression model. The only part missing is the function and gradient calculation inside the *funObj* code. Modify this function to implement the objective function and gradient based on the Huber loss (from the previous section). Hand in your code, as well as the plot obtained using this robust regression approach with $\epsilon = 1$.

Answer: The code is shown in Figure 16, and the plot is shown in Figure 17.

```
function robustRegressionObj(w,X,y)

    (n, d) = size(X)
    f = 0
    g = zeros(size(w))

    # Forward propagation of Huber loss
    for i in 1:n
        # Compute the residual
        ri = w' * X[i, :] .- y[i]
        ri = ri[1]

        if abs(ri) <= 1
            f += 0.5*ri^2
            g += ri .* X[i, :]
        else
            f += abs(ri) - 0.5
            g += sign.(ri) .* X[i, :]
        end
    end

    return (f,g)
end
```

Figure 16: Implementation of Huber forward and back propagation.

[1] Though sometimes the numerical gradient checker itself can be wrong. For a lot more on numerical differentiation you can take CPSC 303.

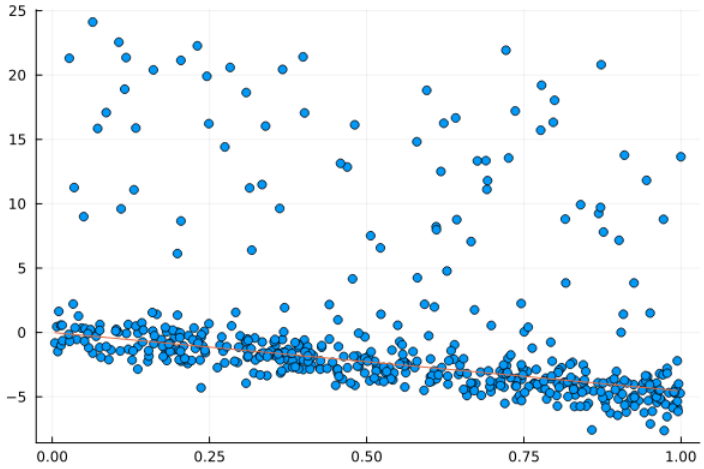


Figure 17: Obtained plot for regression with Huber loss.

5 Very-Short Answer Questions

1. Describe a dataset with k clusters where k-means cannot find the true clusters.

   Answer: K-means partitions the space into convex regions. Therefore, if the data has a non-convex shape, K-means fails to properly cluster the data. For example, (as seen in lecture), if the clusters have banana shapes that are inter-connected together or clusters are cocentric, K-means fails.

2. Why do we need random restarts for k-means but not for density-based clustering?

   Answer: K-means requires k initial cluster means, and it is sensitive to such initialization in a way that the final obtained clusters may differ. Therefore, random initialization/restarts are used and the best clustering is chosen. However, for density-based algorithms, no pre-defined measurements are required and the final clusters remain the same after re-running the model.

3. Why is it not a good idea to create an ensemble out of multiple k-means runs with random restarts and, for each example, output the mode of the label assignments (voting)?

   Answer: For k-means, while the obtained labels can be used to determine which samples belong in the same cluster, the labels themselves do not have a semantic meaning, and each run of the model can cause the labels to be permuted (label-switching). Therefore, by using this voting ensemble model, all labels could be equally likely with a large number of initializations.

4. For each outlier detection method below, list an example method and a problem with identifying outliers using this method:

   - Model-based outlier detection.

     Answer:

     – Example: Finding the z-score of samples after fitting a probabilistic model.

     – Problem: z-score assumes that the data is concentrated around the mean. Therefore, if we have two clusters, the mean of the data would be a point in between these two cluster. Any sample in that region has a very low z-score. However, that point is likely to be an outlier.

   - Graphical-based outlier detection.

     Answer:

- Example: Using a box-plot and allowing a human to decide whether each point is an outlier.
- Problem: Box-plot only allows 1-variable samples to be considered.

- Supervised outlier detection.

  Answer:

  - Example: Direct classification of samples as inlier or outlier using labelled data (for example using decision trees)
  - Problem: New types of outliers may not be detected (if not seen and known during training of the outlier detector)

5. Why do we minimize $\frac{1}{2}\sum_{i=1}^{n}(wx_i - y_i)^2$ instead of the actual mean squared error $\frac{1}{n}\sum_{i=1}^{n}(wx_i - y_i)^2$ in (1D) least squares?

   Answer: Changing the positive scalar that is being multiplied by the sum of squared residuals does not change the minimizer w. However, by choosing $\frac{1}{2}$, the gradient is simplified since $\frac{1}{2} * 2 = 1$.

6. Give an example of a feature matrix X for which the least squares problem *cannot* be solved as $w = (X^\top X)^{-1}(X^\top y)$.

   Answer: It cannot be solved if $X^\top X$ is non-invertible (its determinant is 0). This happens for

$$X = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$$

7. Why do we typically add a column of 1 values to X when we do linear regression? Should we do this if we're using decision trees?

   Answer: Adding a column of 1's adds a bias term to our linear regression formula. This way, the decision line doesn't have to go through the origin. For decision trees, this is not necessary since the model is not limited to the origin and the space is divided into regions based on a decision for each feature.

8. When should we consider using gradient descent to approximate the solution to the least squares problem instead of exactly solving it with the closed form solution?

   Answer: When the feature dimension d is large. This is because the complexity of the closed formula is $O(nd^2)$, whereas the complexity of the gradient-descent approach is $O(ndt)$.

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

   Answer: If a function is convex, its stationary point(s) is at a global minimum. Yes, convexity does imply that a stationary point exists.

10. For robust regression based on the L1-norm error, why can't we just set the gradient to 0 and solve a linear system? In this setting, why we would want to use a smooth approximation to the absolute value?

    Answer: Because for L1-norm, the function is non-differentiable at the center, so we can't just set the gradient to 0. By using a smooth approximation, the function is differentiable everywhere, and we can use gradient descent to find a solution.

11. What is the problem with having too small of a learning rate in gradient descent?

    Answer: By having a small learning rate, the model takes much longer to converge. Additionally, too small of a learning rate can cause the model to get stuck in a local minimum.

12. What is the problem with having too large of a learning rate in gradient descent?

    Answer: Too large of a learning rate can cause the model to overshoot minima, and the loss can start increasing.

CPSC 340 Assignment 4 (due Monday November 14 at 11:55pm)

Name(s) and Student ID(s):

Answer: Masoud Mokhtari, 14186167

1 Gaussian RBFs and Regularization

Unfortunately, in practice we often do not know what basis to use. However, if we have enough data then we can make up for this by using a basis that is flexible enough to model any reasonable function. These may perform poorly if we do not have much data, but can perform almost as well as the optimal basis as the size of the dataset grows. In this question you will explore using Gaussian radial basis functions (RBFs), which have this property. These RBFs depend on a parameter σ, which (like p in the polynomial basis) can be chosen using a validation set. In this question, you will also see how cross-validation allows you to tune parameters of the model on a larger dataset than a strict training/validation split would allow.

1.1 Regularization

If you run the demo *example_RBF.jl*, it will load a dataset and randomly split the training examples into a "train" and a "validation" set (it does this randomly since the data is sorted). It will then search for the best value of σ for the RBF basis. Once it has the "best" value of σ, it re-trains on the entire dataset and reports the training error on the full training set as well as the error on the test set.

A strange behaviour appears: if you run the script more than once it might choose different values of σ. Sometimes it chooses a large value of σ (like 32) that follows the general trend but misses the oscillations. Other times it sets $\sigma = 1$ or $\sigma = 2$, which fits the oscillations better but overfits so achieves a similar test error.[1] Write a function *leastSquaresRBF*, that fits the model with L2-regularization. Hand in your code, and report the test error you obtain if you train on the full dataset with $\sigma = 1$ and $\lambda = 10^{-12}$ (a very small value).

Hint: to construct an identity matrix in Julia, use the linear algebra package (*using LinearAlgebra*) and then use I to make an identity matrix of the appropriate size (Julia figures out the dimensions for you).

Answer: The code is shown in Figures 1 and 2 where I've used the deviations we obtained in the lectures for L2-regularization. As shown in Figure 3, a test error of 71.17 is obtained.

[1] This behaviour seems to be dependent on your exact setup. Because the $Z^T Z$ matrix with the RBF matrix is really-badly behaved numerically, different floating-point and matrix-operation implementations will handle this in different ways: in some settings it will actually regularizer for you!

```
function leastSquaresRBFL2(X,y,sigma,lambda)
    (n,d) = size(X)

    Z = rbf(X,X,sigma)

    w = (Z'*Z + lambda*I)\(Z'*y)

    predict(Xhat) = rbf(Xhat,X,sigma)*w

    return LinearModel(predict,w)
end
```

Figure 1: The RBF Least Squares function implementation with L2 regularization

```
sigma = 1

# Fit the model based on the full dataset
model = leastSquaresRBFL2(X,y,sigma,10^(-12))

# Report the error on the test set
t = size(Xtest,1)
yhat = model.predict(Xtest)
testError = sum((yhat - ytest).^2)/t
@printf("With sigma of %.3f, testError = %.2f\n",sigma,testError)
```

Figure 2: Code to train and test on the dataset using the L2-regularized RBF implementation.

```
julia> include("example_RBF_L2.jl")
With sigma of 1.000, testError = 71.17
```

Figure 3: Test results using the L2-regularized RBF model.

1.2 Cross-Validation

Even with regularization, the randomization of the training/validation sets has an effect on the value of σ that we choose (on some runs it still chooses a large σ value). This variability would be reduced if we had a larger "train" and "validation" set, and one way to simulate this is with *cross-validation*. Modify the training/validation procedure to use 10-fold cross-validation to select σ (with λ fixed at 10^{-12}). Hand in your code and report how this affects the selection of σ compared to the original code.

Answer: The implementation is shown in Figure 4. Using this code, we obtain the lowest validation error for $\sigma = 1$ or $\sigma = 0.5$ and obtain a test error of 71.17. Using cross-validation, the hyper-parameter selection process is more stable and is affected less by the random selection of the validation set. This means that by running the code for multiple times, best σ fluctuates much less to give us the best validation performance as opposed to the original code where based on the selected validation set, we could come up with a largely different σ value each time.

```
# Data is sorted, so randomize the data first
n = size(X,1)
perm = randperm(n)
minErr = Inf
bestSigma = []

for sigma in 2.0.^(-15:15)
    validError = 0

    for fold_num in 1:10
        validStart = Int64(n/10 * (fold_num - 1) + 1)
        validEnd = Int64(n/10 * fold_num)

        validNdx = perm[validStart:validEnd] # Indices of validation examples
        trainNdx = perm[setdiff(1:n,validStart:validEnd)] # Indices of training examples
        Xtrain = X[trainNdx,:]
        ytrain = y[trainNdx]
        Xvalid = X[validNdx,:]
        yvalid = y[validNdx]

        # Train on the training set
        model_sigma = leastSquaresRBFL2(Xtrain,ytrain,sigma,10^(-12))

        # Compute the error on the validation set
        yhat_sigma = model_sigma.predict(Xvalid)
        validError += sum((yhat_sigma - yvalid).^2)/(n/2)
    end

    @printf("With sigma = %.3f, validError = %.2f\n",sigma,validError/10)

    # Keep track of the lowest validation error
    if validError < minErr
        global minErr = validError
        global bestSigma = sigma
    end
end
```

Figure 4: 10-Fold Cross Validation Code for RBF Least Squares

1.3 Cost of Non-Parametric Bases

When dealing with larger datasets, an important issue is the dependence of the computational cost on the number of training examples n and the number of features d.

1. What is the cost in big-O notation of training a linear regression model with Gaussian RBFs on n training examples with d features (for fixed σ and λ)?

   Answer: $O(n^2d + n^3)$

2. What is the cost of classifying t new examples with this model?

   Answer: $O(tnd)$

3. When is it cheaper to train using Gaussian RBFs than using the original linear basis?

   Answer: The original linear basis costs $O(nd^2 + d^3)$, meaning that its cost grows much faster than RBF for large d. Therefore, when $d > n$, training RBFs is cheaper.

4. When is it cheaper to predict using Gaussian RBFs than using the original linear basis?

   Answer: The prediction cost of linear basis is $O(td)$, therefore, RBFs are never cheaper than linear basis for prediction because of the feature construction step.

2 Logistic Regression with Sparse Regularization

If you run the function *example_logistic.jl*, it will:

1. Load a binary classification dataset containing a training and a validation set.
2. "Standardize" the columns of X and add a bias variable.
3. Apply the same transformation to $Xvalidate$.
4. Fit a least squares model, using the sign of $w^T x_i$ to make predictions.
5. Report the number of features selected by the model (number of non-zero regression weights).
6. Report the error on the training and validation sets.

Least squares does ok as a binary classifier on this dataset, but it uses all the features (even though only the prime-numbered features are relevant) and the validation error is above the minimum achievable for this model (which is 1 percent, if you have enough data and know which features are relevant). In this question, you will modify this demo to use the logistic loss and to use different forms of regularization to improve on these aspects.

2.1 Logistic Regression

Instead of least squares, modify the script to use logistic regression. You can use the *logReg.jl* file, which implements the training and prediction function for a logistic regresion classifier (using a version of the *findMin* function that does derivative checking for you and that uses more-clever choices of step-sizes). When you switch to using logistic regression, report how the following quantities change: the training error, validation error, and number of features.

Answer: With logistic regression, the model achieves a training error of 0 and a validation error of 0.082. Therefore, this model has achieved lower training and validation errors compared to the least squares variant. The number of features, however, stays the same because we are not using any regularization.

2.2 L2-Regularization

Make a new function, *logRegL2*, that takes an input parameter λ and fits a logistic regression model with L2-regularization. Specifically, while *logReg* computes w by minimizing

$$f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i)),$$

your new function *logRegL2* should compute w by minimizing

$$f(w) = \sum_{i=1}^{n} \left[\log(1 + \exp(-y_i w^T x_i))\right] + \frac{\lambda}{2}\|w\|^2.$$

Hand in the objective function that your updated code minimizes, and using $\lambda = 1.0$ report how the following quantities change: the training error, the validation error, the number of features used, and the number of gradient descent iterations.

Answer: The code is shown in Figure 5. The model achieves a training error of 0.002, a validation error of 0.074, meaning that training error slightly increases, and validation error decreases. Additionally, we see that the number of features remains the same (101) since we don't expect sparsity from L2 regularization. Lastly, the number gradient descent iterations decreases from 86 to 30.

```
25
26  function logRegL2(X,y,lambda)
27
28      (n,d) = size(X)
29
30      # Initial guess
31      w = zeros(d,1)
32
33      # Function we're going to minimize (and that computes gradient)
34      funObj(w) = logisticObjL2(w,X,y,lambda)
35
36      # Solve least squares problem
37      w = findMin(funObj,w,derivativeCheck=true)
38
39      # Make linear prediction function
40      predict(Xhat) = sign.(Xhat*w)
41
42      # Return model
43      return LinearModel(predict,w)
44  end
45
46  function logisticObjL2(w,X,y,lambda)
47      yXw = y.*(X*w)
48      f = sum(log.(1 .+ exp.(-yXw))) + lambda/2 * norm(w)^2
49      g = -X'*(y./(1 .+ exp.(yXw))) + lambda*w
50      return (f,g)
51  end
```

Figure 5: L2 Regularized Version of Logistic Regression.

2.3 L1-Regularization

Make a new function, *logRegL1*, that takes an input parameter λ and fits a logistic regression model with L1-regularization,

$$f(w) = \sum_{i=1}^{n} \left[\log(1 + \exp(-y_i w^T x_i))\right] + \lambda \|w\|_1.$$

Hand in your *logRegL1* code. Using this new code and $\lambda = 1$, report the following quantities: the training error, the validation error, and the number of features the model uses.

You should use the function *findMinL1*, which implements a proximal-gradient method to minimize the sum of a differentiable function g and $\lambda\|w\|_1$,

$$f(w) = g(w) + \lambda \|w\|_1.$$

This function has a similar interface to *findMin*, except that you (a) only provide the code to compute the function/gradient of the differentiable part g and (b) need to provide the value λ.

Answer: The implementation is shown in Figure 6. We obtain a training error of 0, a validation error of 0.052, and the model uses 71 features, which is the result of the sparsity achieved by L1 regularization.

```
60  function logRegL1(X,y,lambda)
61
62      (n,d) = size(X)
63
64      # Initial guess
65      w = zeros(d,1)
66
67      # Function we're going to minimize (and that computes gradient)
68      funObj(w) = logisticObj(w,X,y)
69
70      # Solve least squares problem
71      w = findMinL1(funObj,w,lambda)
72
73      # Make linear prediction function
74      predict(Xhat) = sign.(Xhat*w)
75
76      # Return model
77      return LinearModel(predict,w)
78  end
```

Figure 6: L1 Regularized Version of Logistic Regression.

2.4 L0-Regularization

The function *logRegL0* contains part of the code needed to implement the *forward selection* algorithm, which approximates the solution with L0-regularization,

$$f(w) = \sum_{i=1}^{n} \left[\log(1 + \exp(-y_i w^T x_i))\right] + \lambda \|w\|_0.$$

The 'for' loop in this function is missing the part where we fit the model using the subset *Sj*, then compute the score and updates the *minScore/minS*. Modify the 'for' loop in this code so that it fits the model using only the features *Sj*, computes the score above using these features, and updates the *minScore/minS* variables (if you want to turn off the diagonistics generated by *findMin*, you can use *verbose* = *false*).[2] Hand in your updated code. Using this new code, set $\lambda = 1$ and report: the training error, the validation error, and the number of features used.

Note that the code differs a bit from what we discussed in class, since we assume that the first feature is the bias variable and assume that the bias variable is always included. Also, note that for this particular case using the L0-norm with $\lambda = 1$ is equivalent to what is known as the Akaike information criterion (BIC) for variable selection.

Answer: The code is shown in Figure 7. A training error of 0, a validation error of 0.018 is achieved. The number of features is also reduced to 24.

```
        # Print out the variables we've selected so far
        @printf("Current set of selected variables (score = %f):\n",minScore)
        for j in 1:length(S)
            @printf("%d ",S[j])
        end
        @printf("\n")

        for j in setdiff(1:d,S)
            # Fit the model with 'j' added to the feature set 'S'
            # then compute the score and update 'minScore' and 'minS'
            Sj = [S;j]
            Xs = X[:,Sj]

            w = zeros(length(Sj),1)
            w = findMin(funObj,w,verbose=false)
            (f,~) = funObj(w)
            score = f + lambda*length(Sj)

            if score < minScore
                minScore = score # Lowest score we've found
                minS = Sj # Best set of features we've found
            end
        end
        S = minS
    end
```

Figure 7: The Modified Portion of Code for L0 Regularized Logistic Regression.

[2]Note that Julia doesn't like when you re-define functions, but if you change the variable *Xs* it will actually change the behaviour of the *funObj* that is already defined.

2.5 L1-Regularization vs. L0-Regularization

For this problem, the relevant features are the bias variable and the featurs with prime numbers. Given this, explain how each of the 3 regularizers (L2-regularization, L1-regularization, and L0-regularization) performed in terms of false positives for feature selection (a false positive would be when a feature is selected but it is not relevant). And then explain how each method did in terms of false negatives..

Answer:

- L2: With L2 regularization, the model uses all features. Therefore, it is showing the maximum number of false positives possible. For the same reason, the model shows no false negatives. In summary, $FP = 74$ and $FN = 0$.
- L1: With L1, although some sparsity is achieved, the model chooses 75 features, which is more than the number of prime number features. Printing the selected features, I saw that there were no false negatives. In summary, $FP = 44$ and $FN = 0$.
- L0: With L0, the number of non-zero features is 24, and looking at the selected features, we see that there are no false positives. However, there are three false negatives including 29, 61, and 97. In summary, $FP = 0$ and $FN = 3$.

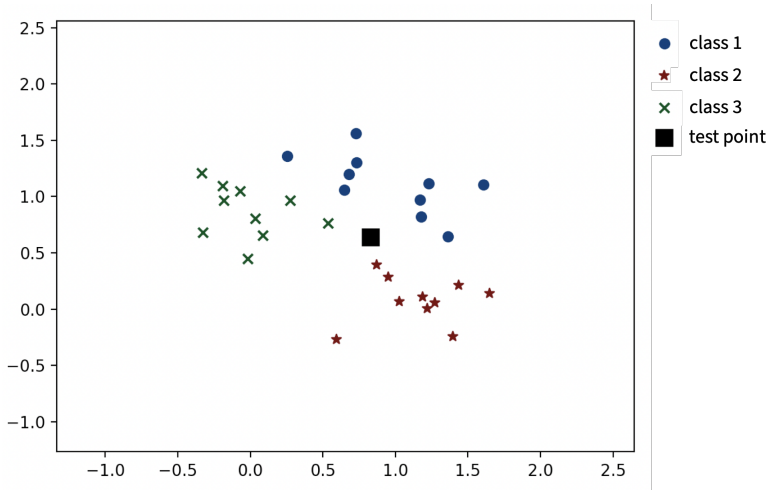
3 Multi-Class Logistic

The function *example_multiClass* loads a multi-class classification dataset with $y_i \in \{1, 2, 3, 4, 5\}$ and fits a 'one-vs-all' classification model using binary logistic regression, 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 that examples will be in class 1 (corresponding to the blue circles).

3.1 Softmax Classification

Linear classifiers make their decisions by finding the class label c maximizing the quantity $w_c^T x_i$, so we want to train the model to make $w_{y_i}^T x_i$ larger than $w_{c'}^T x_i$ for all the classes c' that are not the true label y_i. Here, c is a possible label and $w_{c'}$ is **row** c' of W. Similarly, y_i is the training label, w_{y_i} is **row** y_i of W, and in this setting we are assuming a discrete label $y_i \in \{1, 2, \ldots, k\}$. Before we move on to implementing the softmax classifier to fix the issues raised in the introduction, let's do a simple example:

Consider the dataset below, which has 30 training examples, 2 features, and 3 class labels:



Suppose that we want to classify the black square at the location

$$\hat{x} = \begin{bmatrix} +0.84 \\ +0.64 \end{bmatrix}.$$

Suppose that we fit a multi-class linear classifier including bias variable using the softmax loss. We obtain the weight matrix

$$\begin{array}{c} \\ \mathrm{W} = \end{array} \begin{array}{c} \begin{array}{ccc} x^1 & x^2 & \text{bias} \end{array} \\ \begin{bmatrix} +1.62 & +3.47 & +0.00 \\ -3.83 & +0.67 & +4.90 \\ +2.22 & -4.13 & +3.37 \end{bmatrix} \end{array},$$

where the first two columns correspond to the two features and the *last column* corresponds to the bias variable.

1. What is the meaning of the rows w_i in the matrix W?

   Answer: Each row can be considered a one-vs-all classifier for the corresponding class. In other words, each row, indicates which features are important or unimportant to say a sample belongs to class i.

2. Under this model, what class label would we assign to the test example $\hat{x}$? (show your work)

Answer:

$$\hat{x} = \begin{bmatrix} +0.84 \\ +0.64 \\ 1 \end{bmatrix}$$

$$W\hat{x} = \begin{bmatrix} 0.84 \times 1.62 + 0.64 \times 3.47 \\ 0.84 \times -3.83 + 0.64 \times 0.67 + 4.9 \\ 0.84 \times 2.22 + 0.64 \times -4.13 + 3.37 \end{bmatrix} = \begin{bmatrix} 3.5816 \\ 2.1116 \\ 2.5916 \end{bmatrix}$$

Therefore, since $w_1^T \hat{z}$ get the largest score, the sample is classified as class 1.

3.2 Softmax Loss

Using a one-vs-all classifier hurts performance because the classifiers are fit independently, so there is no attempt to calibrate the **rows** of the matrix W. An alternative to this independent model is to use the softmax loss function, which for n training examples is given by

$$f(W) = \sum_{i=1}^{n}\left[-w_{y_i}^T x_i + \log\left(\sum_{c'=1}^{k}\exp(w_{c'}^T x_i)\right)\right].$$

Derive the partial derivative $\frac{\partial f}{\partial W_{jc}}$ of this loss function with respect to a particular element W_{cj} (the variable in row c and column j of the matrix W). Try to simplify the derivative as much as possible (but you can express the result in summation notation).

Hint: for the gradient you can use x_{ij} to refer to element j of example i. For the first term you will need to separately think about the cases where $c = y_i$ and the cases where $c \neq y_i$. You may find it helpful to use an 'indicator' function, $I(y_i = c)$, which is 1 when $y_i = c$ and is 0 otherwise. Note that you can use the definition of the softmax probability to simplify the second term of the derivative.

Answer: For the first term in the summation, we have:

$$\frac{\partial \sum_{i=1}^{n} -w_{y_i}^T x_i}{\partial W_{jc}} = \sum_{i=1}^{n} -I(y_i = c)x_{ij}$$

For the second term, we have:

$$\frac{\partial \sum_{i=1}^{n} \log\left(\sum_{c'=1}^{k}\exp(w_{c'}^T x_i)\right)}{\partial W_{jc}} = \sum_{i=1}^{n}\frac{1}{\sum_{c'=1}^{k}\exp(w_{c'}^T x_i)}\exp(w_c^T x_i)x_{ij} = \sum_{i=1}^{n}\frac{\exp(w_c^T x_i)}{\sum_{c'=1}^{k}\exp(w_{c'}^T x_i)}x_{ij}$$

Putting these together:

$$\frac{\partial f}{\partial W_{jc}} = \sum_{i=1}^{n} -I(y_i = c)x_{ij} + \frac{\exp(w_c^T x_i)}{\sum_{c'=1}^{k}\exp(w_{c'}^T x_i)}x_{ij}$$

It must be noted that the second term can be interpreted as the probability of x_i being classified as c.

3.3 Softmax Classifier

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 will want to use the *derivativeCheck* option in *findMin.jl* to check that your gradient code is correct. Also, note that *findMin.jl* expects that the parameter vector and gradient are *column vectors*. The easiest way to work around these issues is to use the *reshape* command: call *findMin.jl* with a $dk \times 1$ vector w and at the start of your objective function, reshape w to be a $k \times d$ matrix W, then compute the $k \times d$ matrix of partial derivatives, and finally reshape this to be the $dk \times 1$ gradient vector.

Answer: The code is shown in Figure 8. The classifier achieves a validation error of 0.014. (Please note that I am using random initialization for w, which is not ideal, but I believe it suffices for this question.)

```
function softmaxClassifier(X,y)
    (n,d) = size(X)
    k = maximum(y)

    # Initial guess
    w = randn(d*k,1)

    # Function we're going to minimize (and that computes gradient)
    funObj(w) = softmaxObjective(w,X,y)

    # Solve least squares problem
    w = findMin(funObj,w,derivativeCheck=true)
    w = reshape(w, (k, d))

    # Make linear prediction function
    predict(Xhat) = last.(Tuple.(findmax(Xhat*w', dims=2)[2]))

    # Return model
    return LinearModel(predict,w)
end

function softmaxObjective(w,X,y)
    (n,d) = size(X)
    k = maximum(y)

    w = reshape(w, (k, d))

    f = -sum([(X*w')[i, y[i]] for i in 1:length(y)]) + sum(log.(sum(exp.(X*w'), dims=2)))
    g = zeros(k,d)

    for c in 1:k
        for j in 1:d
            g[c,j] = -sum(X[y.==c, j]) + sum(X[:, j] .* exp.((X*w')[:, c]) ./ sum(exp.(X*w'), dims=2))
        end
    end

    g = reshape(g, (d*k, 1))

    return (f, g)

end
```

Figure 8: Softmax Classifier Implementation

3.4 Cost of Multinomial Logistic Regression

Assuming that we have

- n training examples.
- d features.
- k classes.
- t testing examples.
- T iterations of gradient descent for training.

1. In big-$\mathcal{O}$ notation, what is the cost of training the softmax classifier?

   Answer: If we compute XW^T beforehand (which takes $O(nkd)$), for each element in W (which has $k \times d$ elements), we need O(n) to compute the gradient. Therefore, the final answer becomes: $O(Tnkd)$.

2. In big-$\mathcal{O}$ notation, what is the cost of classifying the test examples?

   Answer: We need to compute $\hat{X}W^T$ where $\hat{X}$ is t*d and W is k*d. Therefore, O(tdk).

4 Very-Short Answer Questions

1. If we fit a linear regression model and then remove all features whose associated weight is small, why is this an ineffective way of performing feature selection?

   Answer: While the weight for individual features may be low, the accumulation of such features together can have a major impact on the final results (Taco Tuesday is one such example as brought up in the lectures). Removing all features with low weights will remove their collective impact on the output. Additionally, there could be cases where all features are given low weights. In such cases, all features will be removed and the model must predicts while disregarding the input samples altogether.

2. Given 3 features $\{f_1, f_2, f_3\}$, provide an argument that illustrates why the forward selection algorithm is not guaranteed to find an optimal subset of features.

   Answer: In forward selection, features are selected one at a time rather than considering all possible combinations. Consider the case where we would like to find the food that causes sickness with f_1 being milk, f_2 being street food, and f_3 being lemon juice. Combining milk with lemon juice is highly likely to make you sick, but eating/drinking them separately is probably OK. Therefore, initially, we could choose street food as the important feature because it is more likely to make you sick compared to having milk or lemon juice on their own. If we were considering all possible combination, we would have chosen the combination of milk and lemon juice to be the most important features here.

3. What is a setting where you would use the L1-loss, and what is a setting where you would use L1-regularization?

   Answer: We use L1 loss when we would like our model to be robust to outliers. L1 regularization is used where would like our model to perform implicit feature selection through learned weight sparsity.

4. Among L0-regularization, L1-regularization, and L2-regularization: which yield convex objectives? Which yield unique solutions? Which yield sparse solutions?

   Answer:

   - Convex objective: L2, L1
   - Unique solution: L2
   - Sparsity: L0 and L1

5. What is the effect of λ in L1-regularization on the sparsity level of the solution? What is the effect of λ on the two parts of the fundamental trade-off?

   Answer: The higher the λ is the higher the levels of sparsity achieved. Therefore, as we increase λ, E_{train} increases and E_{approx} decreases.

6. Suppose you have a feature selection method that tends not generate false positives but has many false negatives (it misses relevant variables). Describe an ensemble method for feature selection that could improve the performance of this method.

   Answer: Since the method does not produce much FPs, it is a good idea to take the union of the features selected by an ensemble of models. To achieve this, we can use our feature selection model on different bootstraps of the dataset and then use the union of features selected by the individual models.

7. How does the hyper-parameter σ affect the shape of the Gaussian RBFs bumps? How does it affect the fundamental tradeoff?

   Answer: Decreasing σ causes the bumps to be sharper and more concentrated at the sample locations, which causes the features to be more localized and less affected by further samples. Therefore, increasing σ causes E_{train} to increase and E_{approx} to decrease.

8. What is the main problem with using least squares to fit a linear model for binary classification?

   Answer: Using least squares for binary classification causes the model to be penalized when it is "too right". For example, when the model has to predict +1, and it predicts a large number like +100, it is penalized a lot even though it is producing a correct guess.

9. Suppose a binary classification dataset has 3 features. If this dataset is "linearly separable", what does this precisely mean in three-dimensional space?

   Answer: This means that we can find a plane that divides the space into two volumes, correctly indicating the binary labels for all samples.

10. Why do we not minimize $\max(0, -y_i w^\top x_i)$ when we fit a binary linear classifier, even though it's a convex approximation to the 0-1 loss?

    Answer: Because this objective has a degenerate solution at $w = 0$ where the minimum loss is achieved. However, this way the model always produces a score of 0.

11. For a linearly-separable binary classification problem, how does an SVM classifier differ from a classifier found using the perceptron algorithm?

    Answer: In this case, while the perceptron find a perfect classifier, it does not necessarily maximize the margin between the decision boundary and the samples of the two classes. However, SVM does achieve this.

12. Which of the following methods produce linear classifiers? (a) binary least squares as in Question 3, (b) the perceptron algorithm, (c) SVMs, (d) logistic regression, and (e) KNN with $k = 1$ and $n = 2$.

    Answer: All of the above except for KNN produce linear classifiers (Please note that while in this case (n=2, k=1) KNN looks like a linear classifier, that doesn't make KNNs a linear classifier in general.)

13. Why do we use the polynomial kernel to implement the polynomial basis when d and p (degree of polynomial) are large?

    Answer: With large d and p, we have to store a large feature matrix (with $O(d^p)$), which is intractable. Kernel tricks address this issue by allowing us to create more tractable matrices that are intermediate representation of the features matrix such as ZZ^T.

14. What is the relationship between the softmax loss and the softmax function?

    Answer: Softmax function characterizes a probability with $\frac{e^{z_y}}{\sum_{c=1}^{k} e^{z_c}}$. Whereas softmax loss is the negative logarithm of these probabilities such that $-log(\frac{e^{z_y}}{\sum_{c=1}^{k} e^{z_c}}) = -z_y + \sum_{c=1}^{k} e^{z_c}$.

Project Proposal (OPTIONAL FOR 340 STUDENTS)

For 532M students, there is a project component to the course that will be worth 20% of your final grade. For 340 students, there is no requirement to do a project. However, 340 students have the option to do a project anyway for the possibility of obtaining a higher grade: your project grade can replace either your 2 lowest assignment scores or your midterm score (whicheve helps you more).[3]

These projects are done in groups of 2-3. The final deliverable will be a 6-page report that is due near the end of the exam period (something like December 22nd, minus a few days so we have time to grade). It is expected that this project will be a literature survey, but research projects are also ok.

There aren't really any restrictions on the group compositions: 340 students can work with 532M students, auditors can work with registered students, and you can combine this project with a project from another one of your classes (assuming you get the other instructor's permission, and even if not all students in the other class are registered in this class). The only combinations I really want to avoid are having students do projects with the TAs (due to the obvious conflict of interest), project groups that have no students enrolled in 340 or 532M, or projects that contain people taking no CPSC classes.

If you are in 532M, or in 340 and want to do a project, for the final part of this assignment you must a submit a project proposal for your course 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 instructors and the TAs, so you don't need to introduce any ML background but you will need to introduce non-ML topics.

You should submit this question as a group on Gradescope, separate from the other assignment questions.

There is quite a bit of flexibility in terms of the type of project you do, as I believe there are many ways that people can make valuable contributions to research. However, note that ultimately the final deliverable for the project will be a report that emphasizes a particular "contribution" (i.e., what doing the project has added to the world). The reason for this, even though it's strange for some possible projects, is that this is the standard way that results are communicated to the research community.

The three mains ingredients of the project proposal are:

1. What problem you are focusing on.
2. What you plan to do.
3. What will be the "contribution".

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

I encourage you to follow the following default "template" for the project:

1. **Literature review**: you pick a specific topic in ML, read at least 10 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.

The advantage of the above template is that the project will take a somewhat-predictable amount of time. If you want to explore a different style of project, here are some standard "templates":

2. **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

[3] The course is *not* graded on a curve, so 340 students are not hurt by choosing the to skip the project.

showing that some methods work better than others for this specific application (or your contribution could be that everything works equally well/badly).

3. **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.
4. **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. 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.
5. **Improving performance**: you pick a specific machine learning technique, and try to extend it in some way to improve its performance. In this case, the contribution would be the new technique and an evaluation of its performance.
6. **Generalization to new setting**: you pick a specific machine learning technique, and try to extend it to a new setting (for example, making a multi-label 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.
7. **Coding project**: you pick a specific method or set of methods, 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.
8. **Theory**: you pick a theoretical topic (like the variance of cross-validation), 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.
9. **Reproduction**: you take a recent paper published in one of the top machine learning venue (either NeurIPS, ICML, ICLR, AI/Stats, or JMLR), and try to reproduce the results in the paper.

The above are just suggestions, and projects could 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 (and again I recommend doing the literature review). Also note that the project can focus on topics not covered in the course (like GANs), so there is flexibility in the topic, but the topic should be closely-related to ML.

This question is mandatory but will not be formally marked: it is just a sanity check that you have at least one project idea that has an appropriate topic and scope, that you find a group early, and that you allocate some time to thinking about the project. Also, 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 (and changing groups is ok too). If you aren't sure what to do, go bug the TAs in office hours (which is a good idea even if you are sure what you want to).

CPSC 340 Assignment 5 (due Friday November 25 at 11:55pm)

Name(s) and Student ID(s):

Answer: Masoud Mokhtari, 14186167

1 Kernel Trick and SGD

In this question you will revisit questions from previous assignments, this time implementing the same (or similar) models using the "kernel trick" and SGD.

1.1 "Other" Normal Equations

The script *example_nonLinear* loads a dataset from a previous assignment, and fits an L2-regularized least squares model with a bias term (the regularization leads to a small improvement in the test error, even for this 2-variable problem). Modify the *leastSquaresBiasL2* function so that it uses the "other" normal equations we discussed in class,

$$v = Z^T \underbrace{(ZZ^T + \lambda I)^{-1} y}_{u}.$$

In particular, the training should result in an $n \times 1$ vetor u of parameters, and predictions are made based on the training Z and the vector u. Hand in your code for the modified function.

Hint: you should get the same predictions as the original funciton. To help debugging, you may want to first re-write the calculation of v using the above formula, to verify that this gives you the same vector v.

Answer: The implementation is shown in Fig 1.

```
32  function leastSquaresBiasL2Kernel(X,y,lambda)
33
34      # Add bias column
35      n = size(X,1)
36      Z = [ones(n,1) X]
37
38      # Find regression weights minimizing squared error
39      u = (Z*Z' + lambda*I) \ y
40
41      # Make linear prediction function
42      predict(Xhat) = [ones(size(Xhat,1),1) Xhat]*Z'*u
43
44      # Return model
45      return LinearModel(predict,u)
46  end
```

Figure 1: Least Squares using the kernel trick.

1.2 Polynomial Kernel

Write a new function, *leastSquaresKernelBasis*, taking a degree p and a regularization parameter λ. It should fit a degree-p polynomial to the data, using the kernel trick to avoid ever forming the matrix Z. Hand in your code and the plot obtained with $p = 3$ and $\lambda = 10^{-6}$.

Hint: you may find it helpful to write a function *polyKernel* that takes two matrices as inputs (either X and X or $\tilde{X}$ and X) and a degree p, and computes the polynomial kernel between all pairs of rows in the matrices.

Answer: The code and plot are shown in Figs 4 and 5.

```
function leastSquaresKernelBasis(X,y, p, lambda)

    K = polyKernel(X, X, p)

    # Find regression weights minimizing squared error
    u = (K + lambda*I) \ y

    # Make linear prediction function
    function predict(Xhat)
        Khat = polyKernel(Xhat, X, p)
        return Khat*u
    end

    # Return model
    return LinearModel(predict,u)
end

function polyKernel(A, B, p)
    t = size(A, 1)
    n = size(B, 1)

    K = zeros(t, n)

    for row in 1:t
        for col in 1:n
            K[row, col] = (1 + A[row, :]' * B[col, :]) ^ p
        end
    end

    return K
end
```

Figure 2: Implementation for the polynomial least squares using the kernel trick.

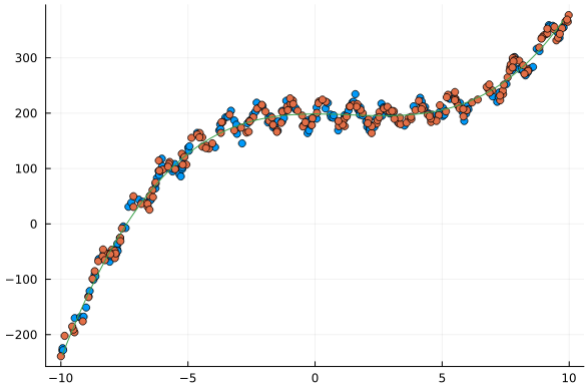


Figure 3: Plot using polynomial least squares with the kernel trick.

1.3 Gaussian-RBF Kernel

Repeat the previous question and report the same quantities, but using the Gaussian RBF kernel. You can use $\sigma = 1$ and $\lambda = 10^{-6}$ for the plot.

Answer: The code and plot are shown in Figs 4 and 5.

```
function leastSquaresRBFKernelBasis(X,y, sigma, lambda)

    K = RBFKernel(X, X, sigma)

    # Find regression weights minimizing squared error
    u = (K + lambda*I) \ y

    # Make linear prediction function
    function predict(Xhat)
        Khat = RBFKernel(Xhat, X, sigma)
        return Khat*u
    end

    # Return model
    return LinearModel(predict,u)
end

function RBFKernel(A, B, sigma)
    t = size(A, 1)
    n = size(B, 1)

    K = zeros(t, n)

    for row in 1:t
        for col in 1:n
            K[row, col] = exp(-norm(A[row, :] - B[col, :])^2 / (2* sigma^2))
        end
    end

    return K
end
```

Figure 4: Implementation for the RBF least squares using the kernel trick.

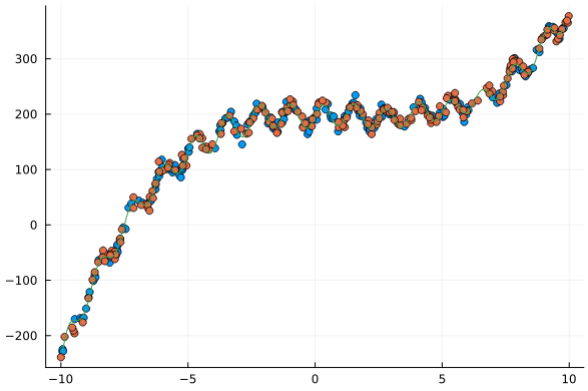


Figure 5: Plot using RBF least squares with the kernel trick.

1.4 Stochastic Gradient Descent

Instead of using normal equations to fit a linear least squares model, the script *example_GD* fits the regularized linear model using gradient descent. It uses 500 iterations and a step size that guarantees that the regularized squared error decreases on each step. Modify this demo to implement stochastic gradient descent update, which on iteration t uses

$$v^{t+1} = v^t - \alpha_t((v^T z_i - y_i)z_i + (\lambda/n)v),$$

where on each iteration i is a random training example.

1. Hand in your code implementing the SGD update.

   Answer: In Fig 6, I show the modified portion of the code.

```
# Fit least squares with bias via gradient descent
n = size(X,1)
Z = [ones(n,1) X]
lambda = 1
v = zeros(2,1)
alpha = 1/(norm(Z,2)^2 + lambda)

for t in 1:500

    # Randomly choose i
    i = rand(1:n)

    global v -= alpha*((v'*(Z[i, :]) - y[i, :]).*Z[i, :] + (lambda/n)*v)

end

predict(Xhat) = [ones(size(Xhat,1),1) Xhat]*v
model = LinearModel(predict,v)
```

Figure 6: Modified portion of the code for SGD.

2. Consider step sizes of the form $\alpha_t = \gamma/t$, $\alpha_t = \gamma/\sqrt{t}$, and $\alpha_t = \gamma$ (for some constant γ). If we consider value of γ that are powers of 10, and run SGD for 500 iterations with $\lambda = 1$, what value tends to give the best performance for each of these types of step sizes?

   Answer:

   - $\alpha_t = \gamma/t$: We get the best performance by using $\gamma = 10^{-1}$.
   - $\alpha_t = \gamma/\sqrt{t}$: We get the best performance by using $\gamma = 10^{-1}$.
   - $\alpha_t = \gamma$: We get the best performance by using $\gamma = 10^{-2}$.

   We show the loss trends in Figs 7, 8, and 9.

```
gamma/t with power = -3.000000 Squared train Error with least squares: 35573.904
gamma/t power = -3.000000 Squared test Error with least squares: 39471.162
gamma/t with power = -2.000000 Squared train Error with least squares: 25690.523
gamma/t power = -2.000000 Squared test Error with least squares: 26248.874
gamma/t with power = -1.000000 Squared train Error with least squares: 10427.902
gamma/t power = -1.000000 Squared test Error with least squares: 10302.316
gamma/t with power = 0.000000 Squared train Error with least squares: 5070031860.569
gamma/t power = 0.000000 Squared test Error with least squares: 5594841399.355
```

Figure 7: Loss trend using $\alpha_t = \gamma/t$.

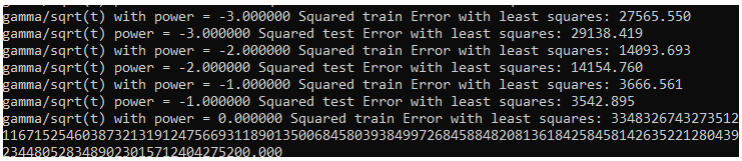


Figure 8: Loss trend using $\alpha_t = \gamma/\sqrt{t}$.

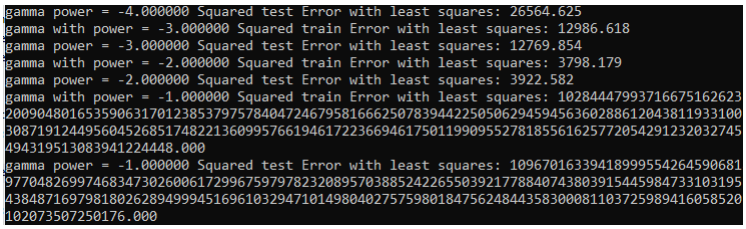


Figure 9: Loss trend using $\alpha_t = \gamma$.

3. On this problem, which of the three types of step size do you think is best to use? And which is the worst to use?

   Answer: I think the best schedule to use is $\alpha_t = \gamma/\sqrt{t}$, and the worst one is $\alpha_t = \gamma$. This is because using $\alpha_t = \gamma$ ignores the fact that the step size must be reduced when we get closer to the minimum and the model keeps overshooting around this point. On the other hand, using $\alpha_t = \gamma/t$ might cause the step size to drop too quickly.

2 MAP Estimation

In class, we considered MAP estimation in a regression model where we assumed that:

- The likelihood $p(y_i \mid x_i, w)$ for each example i is a normal distribution with a mean of $w^T x_i$ and a variance of 1.
- The prior $p(w_j)$ for each variable j is a normal distribution with a mean of zero and a variance of λ^{-1}.

Under these assumptions we showed that computing the MAP estimate with n training examples leads to the standard L2-regularized least squares objective function:

$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2.$$

For each of the alternate assumptions below, write down the objective function that results (from minimizing the negative log-posterior, and simplifying as much as possible):

1. We use a Laplace likelihood with a mean of $w^T x_i$ and a scale of 1, and we use a zero-mean Laplace prior for each variable with a scale parameter of λ^{-1},

$$p(y_i \mid x_i, w) = \frac{1}{2}\exp(-|w^T x_i - y_i|), \quad p(w_j) = \frac{\lambda}{2}\exp(-\lambda|w_j|).$$

Answer: $p(w \mid D) \propto p(y|w, X)p(w)$. Therefore, using the independence assumption, we have:

$$p(y|w, X)p(w) = \prod_{i=1}^{n}[p(y_i \mid x_i, w)] \prod_{j=1}^{d} p(w_j)$$

$$p(y|w, X)p(w) = \prod_{i=1}^{n}[\frac{1}{2}\exp(-|w^T x_i - y_i|)] * \prod_{j=1}^{d}[\frac{\lambda}{2}\exp(-\lambda|w_j|)]$$

Taking the negative logarithm of the above expression, we get:

$$-log(p(y|w, X)p(w)) = -\sum_{i=1}^{n}[log(\frac{1}{2}\exp(-|w^T x_i - y_i|))] - \sum_{j=1}^{d}[log(\frac{\lambda}{2}\exp(-\lambda|w_j|))]$$

$$-log(p(y|w, X)p(w)) = -\sum_{i=1}^{n}[((\text{constant})) - |w^T x_i - y_i|] - \sum_{j=1}^{d}[\text{constant} - \lambda|w_j|]$$

$$-log(p(y|w, X)p(w)) = \sum_{i=1}^{n}[|w^T x_i - y_i|] + \lambda\sum_{j=1}^{d}[|w_j|] + (\text{constant})$$

$$-log(p(y|w, X)p(w)) = ||Xw - y||_1 + \lambda||w||_1 + (\text{constant})$$

Therefore, the objective function becomes:

$$f(w) = ||Xw - y||_1 + \lambda||w||_1$$

2. We use a normal likelihood with a mean of $w^T x_i$ but where each example i has its own positive variance σ_i^2, and a normal prior with a variance of λ^{-1} and a mean that is some "guess" w^0 of the optimal parameter vector,

$$p(y_i \mid x_i, w) = \frac{1}{\sqrt{2\sigma_i^2\pi}}\exp\left(-\frac{(w^T x_i - y_i)^2}{2\sigma_i^2}\right), \quad p(w_j) \propto \exp\left(-\frac{\lambda(w_j - w_j^0)^2}{2}\right).$$

The standard notation for this case is to use Σ as a diagonal matrix with the σ_i^2 values along the diagonal.

Answer: To avoid repetition, I skip the initial multiplication notation and skip to negative log notation:

$$-log(p(y|w,X)p(w)) = -\sum_{i=1}^{n}[log(\frac{1}{\sqrt{2\sigma_i^2\pi}}\exp\left(-\frac{(w^Tx_i-y_i)^2}{2\sigma_i^2}\right))] - \sum_{j=1}^{d}[log(\exp\left(-\frac{\lambda(w_j-w_j^0)^2}{2}\right))]$$

$$-log(p(y|w,X)p(w)) = -\sum_{i=1}^{n}[(\text{constant}) + \left(-\frac{(w^Tx_i-y_i)^2}{2\sigma_i^2}\right)] - \sum_{j=1}^{d}[\left(-\frac{\lambda(w_j-w_j^0)^2}{2}\right)]$$

$$-log(p(y|w,X)p(w)) = (\text{constant}) + \sum_{i=1}^{n}[\frac{(w^Tx_i-y_i)^2}{2\sigma_i^2}] + \frac{\lambda}{2}\sum_{j=1}^{d}[(w_j-w_j^0)^2]$$

$$-log(p(y|w,X)p(w)) = (\text{constant}) + \frac{1}{2}(XW-y)^T\Sigma^{-1}(XW-y) + \frac{\lambda}{2}||w-w^0||_2^2$$

Therefore, the objective function is:

$$f(w) = \frac{1}{2}(XW-y)^T\Sigma^{-1}(XW-y) + \frac{\lambda}{2}||w-w^0||_2^2$$

3. We use a Poisson likelihood with a mean of $\exp(w^Tx_i)$,[1] and we use a uniform prior for some constant κ,

$$p(y_i \mid x_i, w) = \frac{\exp(y_iw^Tx_i)\exp(-\exp(w^Tx_i))}{y_i!}, \quad p(w_j) \propto \kappa$$

For this sub-question you don't need to put likelihood in matrix notation.

Answer:

$$-log(p(y|w,X)p(w)) = -\sum_{i=1}^{n}[log(\frac{\exp(y_iw^Tx_i)\exp(-\exp(w^Tx_i))}{y_i!})] - \sum_{j=1}^{d}[log(\kappa)]$$

$$-log(p(y|w,X)p(w)) = -\sum_{i=1}^{n}[log(\exp(y_iw^Tx_i)) + log(\exp(-\exp(w^Tx_i)) - log(y_i!))] - (\text{constant})$$

$$-log(p(y|w,X)p(w)) = -\sum_{i=1}^{n}[(y_iw^Tx_i) + -\exp(w^Tx_i) - log(y_i!))] - (\text{constant})$$

Therefore, the objective becomes:

$$-\sum_{i=1}^{n}[(y_iw^Tx_i) - \exp(w^Tx_i) - log(y_i!))]$$

Please note that $log(y_i!)$ could also be written as $\sum_{k=1}^{y_i} log(k)$.

4. We use a Laplace likelihood with a mean of w^Tx_i where each example i has its own positive scale paramater v_i^{-1}, and a student t prior (which is very robust to irrelevant features) with ν degrees of freedom,

$$p(y_i \mid x_i, w) = \frac{1}{2}\exp\left(-v_i|w^Tx_i - y_i|\right), \quad p(w_j) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)}\left(1+\frac{w_j^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

[1] This is one way to use regression to model *counts*, like "number of Facebook likes".

where you use can V as a diagonal matrix with the v_i along the diagonal and Γ is the "gamma" function (which is always non-negative). You do not need to put the log-prior in matrix notation.

Answer:

$$-log(p(y|w,X)p(w)) = -\sum_{i=1}^{n}[log(\frac{1}{2}\exp\left(-v_i|w^Tx_i - y_i|\right))] - \sum_{j=1}^{d}[log(\frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)}\left(1+\frac{w_j^2}{\nu}\right)^{-\frac{\nu+1}{2}})]$$

$$-log(p(y|w,X)p(w)) = -\sum_{i=1}^{n}[(\text{constant}) - v_i|w^Tx_i - y_i|] - \sum_{j=1}^{d}[(\text{constant}) + log(\left(1+\frac{w_j^2}{\nu}\right)^{-\frac{\nu+1}{2}})]$$

$$-log(p(y|w,X)p(w)) = (\text{constant}) + \sum_{i=1}^{n}[|v_iw^Tx_i - v_iy_i|] + \sum_{j=1}^{d}[\frac{\nu+1}{2}log\left(1+\frac{w_j^2}{\nu}\right)]$$

$$-log(p(y|w,X)p(w)) = (\text{constant}) + ||VXw - Vy||_1 + \frac{\nu+1}{2}\sum_{j=1}^{d}log\left(1+\frac{w_j^2}{\nu}\right)$$

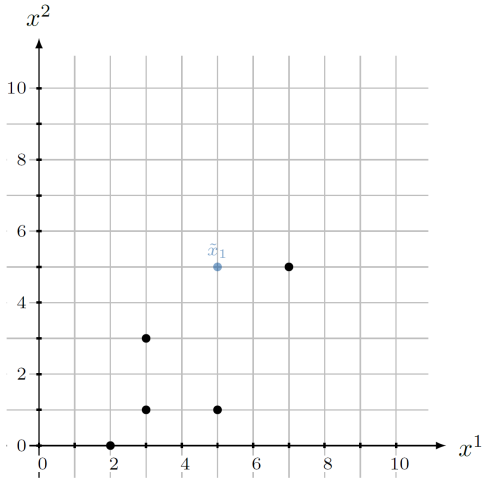
Where we moved v inside the absolute value because v is non-negative. finally, the objective function is:

$$||VXw - Vy||_1 + \frac{\nu+1}{2}\sum_{j=1}^{d}log\left(1+\frac{w_j^2}{\nu}\right)$$

3 Principal Component Analysis (PCA)

3.1 PCA by Hand

Consider the following dataset, containing $n = 5$ training examples and $t = 1$ test example with $d = 2$ features each:



	x^1	x^2
	2	0
	3	1
X	7	5
	3	3
	5	1
$\widetilde{X}$	5	5

1. Principal component analysis requires a centered feature matrix. Find the vector of means $\mu = \begin{bmatrix}\mu_1 \\ \mu_2\end{bmatrix}$ and compute the centered feature matrix X_c.

   Answer: We need to find the column-wise means:

$$\mu_1 = \frac{2+3+7+3+5}{5} = 4$$
$$\mu_2 = \frac{0+1+5+3+1}{5} = 2$$
$$\rightarrow \mu = \begin{bmatrix}4 \\ 2\end{bmatrix}$$

   The normalized version of X becomes:

$$X_c = \begin{bmatrix}2-4 & 0-2 \\ 3-4 & 1-2 \\ 7-4 & 5-2 \\ 3-4 & 3-2 \\ 5-4 & 1-2\end{bmatrix} = \begin{bmatrix}-2 & -2 \\ -1 & -1 \\ 3 & 3 \\ -1 & 1 \\ 1 & -1\end{bmatrix}$$

2. Writing $G_c = \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix} = X_c^\top X_c$, the first principal component $w_1 = \begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix}$ is given by the **normalized** solution to the linear system $(G_c - \mathbf{I}_d\lambda_1)\cdot w_1 = 0$, where λ_1 is the largest solution to the quadratic equation $\chi(\lambda) = (\lambda - g_{11})(\lambda - g_{22}) - g_{12}g_{21} = 0$.[2] The second principal component $w_2 = \begin{bmatrix} w_{21} \\ w_{22} \end{bmatrix}$ is given by the **normalized** solution to the linear equation $w_1^\top w_2 = 0$. Compute the two principal components w_1 and w_2 (show your work).

   Answer: Let's find $X_c^T X_c$ first:

$$X_c^T X_c = \begin{bmatrix} (-2)^2 + (-1)^2 + 3^2 + (-1)^2 + 1^2 & (-2)^2 + (-1)^2 + 3^2 + -1*1 + 1*-1 \\ (-2)^2 + (-1)^2 + 3^2 + -1*1 + 1*-1 & (-2)^2 + (-1)^2 + 3^2 + 1^2 + (-1)^2 \end{bmatrix}$$
$$X_c^T X_c = \begin{bmatrix} 16 & 12 \\ 12 & 16 \end{bmatrix}$$

   We can now find λ_1:

$$\begin{aligned} (\lambda - 16)(\lambda - 16) - 12^2 &= 0 \\ (\lambda - 16)^2 - 12^2 &= 0 \\ |\lambda - 16| &= 12 \\ \lambda &= 4, 28 \end{aligned}$$

   Therefore, $\lambda_1 = 28$. We can now find, w_1:

$$\begin{bmatrix} -12 & 12 \\ 12 & -12 \end{bmatrix} . w_1 = 0$$
$$\begin{bmatrix} -12w_{11} + 12w_{12} \\ 12w_{11} - 12w_{12} \end{bmatrix} = 0$$
$$w_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

   Let's find w_2 now:

$$\frac{1}{\sqrt{2}} w_{21} + \frac{1}{\sqrt{2}} w_{22} = 0$$
$$w_2 = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

   It must be noted that $w_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$ is also another solution (which doesn't make a difference as long as we are consistent).

3. The two principal components w_1 and w_2 form an orthonormal basis $W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$ of $\mathbb{R}^2$. Represent the test point $\tilde{x}_1$ in this basis by performing an orthogonal projection $\tilde{z}_1 = \begin{bmatrix} \tilde{z}_{11} \\ \tilde{z}_{12} \end{bmatrix} = W(\tilde{x}_1 - \mu)$ onto w_1 and w_2 (show your work).

[2] χ is called the *characteristic polynomial* of G_c and its roots are eigenvalues λ whose corresponding eigenvectors w solve the equation $G_c w = \lambda w$.

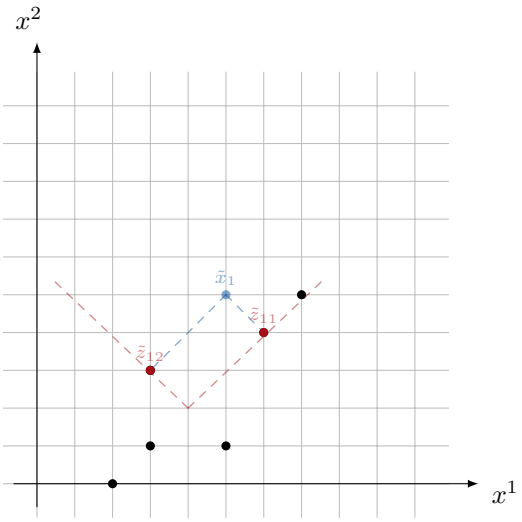
Answer:

$$\tilde{z_1} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 5-4 \\ 5-2 \end{bmatrix}$$

$$\tilde{z_1} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} \frac{4}{\sqrt{2}} \\ \frac{2}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} 2\sqrt{2} \\ \sqrt{2} \end{bmatrix}$$

4. We can now represent the PCA coordinate system in the orginal coordinate system. In the plot above, draw the two coordinate axes $5w_1$ and $5w_2$, using μ as origin. Then visualize the PCA coordinates $\tilde{z}_1$ computed in 3.1.3 by drawing two lines between the test point and its orthogonal projections onto w_1 (given by $\begin{bmatrix} \tilde{z}_{11} \\ 0 \end{bmatrix}$ in the PCA coordinate system) and w_2 (given by $\begin{bmatrix} 0 \\ \tilde{z}_{12} \end{bmatrix}$ in the PCA coordinate system). Finally, use the visualization to read off the coordinates of $\begin{bmatrix} \tilde{z}_{11} \\ 0 \end{bmatrix}$ and $\begin{bmatrix} 0 \\ \tilde{z}_{12} \end{bmatrix}$ in the **original** coordinate system, which can be interpreted as reconstructions of $\tilde{x}_1$. Report both reconstructions and their L2 reconstruction errors (show your work).

   Note: If you want, you can draw the PCA coordinate system and the orthogonal projections directly in Latex; see the commented lines below this one in a5.tex for an example. Hand-drawn results will also be accepted.



Answer: As shown above, the achieved points for $\begin{bmatrix} \tilde{z_{11}} \\ 0 \end{bmatrix}$ and $\begin{bmatrix} 0 \\ \tilde{z_{12}} \end{bmatrix}$ in the original coordinate system are $(6, 4)$ and $(3, 3)$, respectively. Let's find the L2 errors:

$$w_1\text{error} = \sqrt{(6-5)^2 + (4-5)^2} = \sqrt{2}$$
$$w_2\text{error} = \sqrt{(3-5)^2 + (3-5)^2} = \sqrt{8} = 2\sqrt{2}$$

3.2 Data Visualization

The script *example_PCA* will load a dataset containing 50 examples, each representing an animal. The 85 features are traits of these animals. The script also shows a heatmap of the data matrix X. Although it possible to try to interpret this heatmap, it is not as interpretable as a scatterplot.

The function *PCA* applies the classic PCA method (orthogonal bases via SVD) for a given k. Using this function, modify the demo to output a scatterplot of the latent features z_i from the PCA model with $k = 2$. You can use the *scatter* function from *Plots.jl* to output a scatterplot. Use the *annotate!* function to label the points in the scatter plot with the animal names (I set *annotationfontsize=8* when calling this function).

1. Hand in your modified demo and the scatterplot which has the annotated animal names.

   Answer: The code is shown in Fig 10 and the plot is shown in Fig 11.

```julia
using DelimitedFiles
include("PCA.jl")

# Load data
dataTable = readdlm("animals.csv",',')
X = float(real(dataTable[2:end,2:end]))
animal_names = dataTable[2:end, 1]
(n,d) = size(X)

# Plot matrix as image
using Plots
# heatmap(X)

# Use PCA to shrink feature dimension to 2 (k=2)
pca_model = PCA(X, 2)

# Compress the data
X_pca = pca_model.compress(X)

# Plot the scatter plot
scatter(X_pca[:, 1], X_pca[:, 2], legend=false)
title!("Scatter Plot of PCA-Compressed Data")
annotate!(X_pca[:, 1], X_pca[:, 2].-0.1, animal_names, annotationfontsize=8)
```

Figure 10: Modified PCA Demo Code

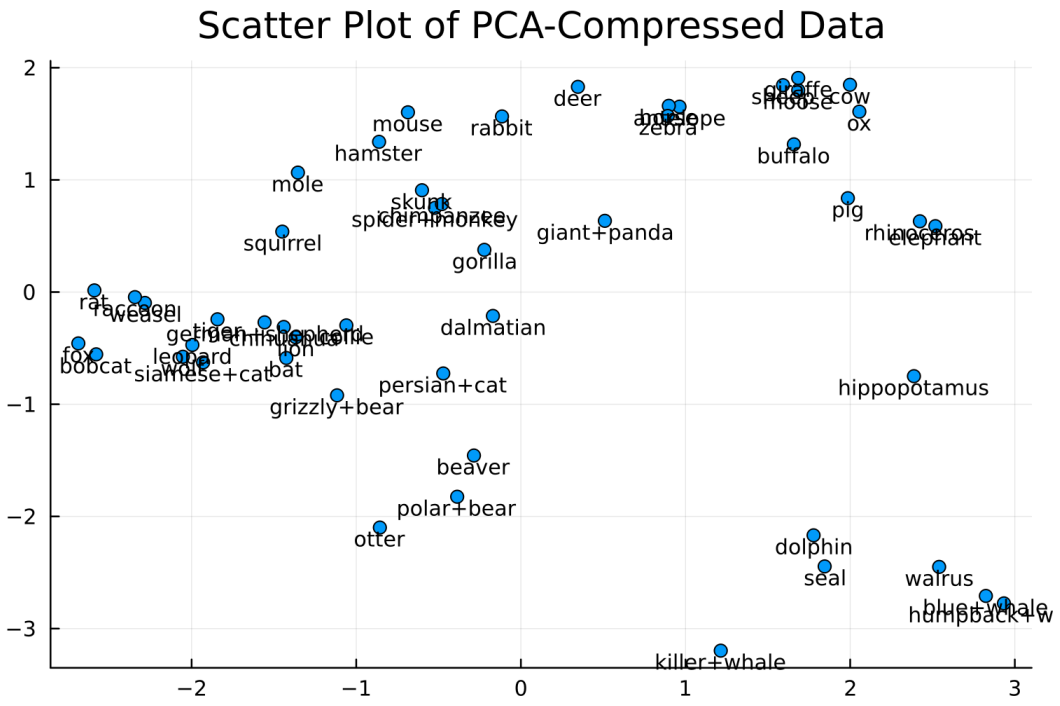


Figure 11: Scatter Plot of PCA Compressed Data.

2. For each of the first three principal components answer the following: which animals are furthest apart according to this principal component, and which trait of the animals has the largest influence (absolute value) on the first principal component? (Make sure not to forget the "+1" when looking for the name of the animal/trait in the *dataTable*).

   Answer:

   - PC1: Farthest animals are "fox" and "humpback whale". "toughskin" has the largest effect.
   - PC2: Farthest animals are "killer whale" and "giraffe". "vegetation" has the largest effect.
   - PC3: Farthest animals are "leopard" and "hamster". "small" has the largest effect.

```
# Compute feature means and center X
X_c = X - repeat(feat_means,n,1)

# Find explained variance
explained_var = 1 - (norm(pca_model.expand(X_pca) - repeat(feat_means,n,1) - X_c)^2 / norm(X_c)^2)
print("The explained variance is:  \n", explained_var * 100)

```

Figure 12: Code computing explained variance.

3.3 Data Compression

It is important to know how much of the information in our dataset is captured by the low-dimensional PCA representation. In class we discussed the "analysis" view that PCA maximizes the variance that is explained by the PCs, and the connection between the Frobenius norm and the variance of a centered data matrix X. Use this connection to answer the following:

1. How much of the variance is explained by our two-dimensional representation from the previous question?

   Answer: Using the code shown in Fig 12, we find that with k=2, only 32.313% of variance is explained.

2. How many PCs are required to explain 75% of the variance in the data?

   Answer: Using the code in Fig 12, we tweak the k value and see that $k = 12$ achieves an explained variance of 75.853%.

Note: you can compute the Frobenius norm of a matrix using the function *norm*. Also, note that the "variance explained" formula from class assumes that X is already centered.

4 Very-Short Answer Questions

1. What is the difference between multi-label and multi-class classification?

   Answer: For multi-class classification, we are categorizing samples into multiple classes (Choosing one class among k classes for a particular sample). In multi-label classification, each sample is assigned multiple labels (for example, dog + brown + well-mannered).

2. We discussed "global" vs. "local" features for e-mail classification. What is an advantage of using global features, and what is advantage of using local features?

   Answer: An advantage of global features is that they can be used for users not known beforehand (generic user) to make a meaningful (although not customized) prediction. The local features personalize the predictions to each user.

3. Describe the kernel trick in L2-regularized least square in one sentence.

   Answer: Kernel trick enables performing L2-regularized least squares tasks in an efficient manner when $n < k$ through the computation of more tractable intermediate features with complexity depending more on n using the "other" normal equations.

4. What is the key advantage of stochastic gradient methods over gradient descent methods?

   Answer: For very large number of samples (n), it is not tractable to perform gradient descent. SGD remedies this by allowing single samples to be used per optimization iteration.

5. Which of the following step-size sequences lead to convergence of stochastic gradient to a stationary point?

   (a) $\alpha^t = 1/t^2$.

   Answer: It converges to a stationary point.

   (b) $\alpha^t = 1/t$.

   Answer: It converges to a stationary point.

   (c) $\alpha^t = \gamma/t$ (for $\gamma > 0$).

   Answer: It converges to a stationary point.

   (d) $\alpha^t = 1/(t + \gamma)$ (for $\gamma > 0$).

   Answer: It converges to a stationary point.

   (e) $\alpha^t = 1/\sqrt{t}$.

   Answer: It converges to a stationary point.

   (f) $\alpha^t = 1$.

   Answer: It does not converge to a stationary point.

6. Given discrete data D and parameters w, is the expression $p(D|w)$ a probability mass function, a likelihood function, or both? Briefly justify your answer.

   Answer: It is both because this probability is characterizing the likelihood of seeing D under parameters w. It must be noted that the area under curve showing $p(D|w)$ for different w does not add up to 1, but that doesn't change the definition of $p(D|w)$ because here we are showing the disribution of D and not w.

7. Why is it often more convenient to minimize the negative log-likelihood than to maximize the likelihood?

Answer: With our independence assumption, our likelihood becomes the multiplication of multiple factors. By using negative log-likelihood, we make computations easier by turning this product of probability over IID examples into sum over examples.

8. How does the impact of the prior in MAP estimation change as we add more data?

   Answer: As more and more data is observed by the model, the impact of our prior knowledge (represented by the prior distribution in MAP) is lessened. Another intuition is that with more data, the effects of overfitting (which the prior is trying to combat) is reduced.

9. What is the difference between a generative model and a discriminative model?

   Answer: A discriminative model tries to maximize $p(y|X, w)$ without having to model X's distribution (conditioned on X being fixed). On the other hand, generative models try to optimize $p(y, X|w)$ where X is also modelled.

10. With PCA, is it possible for the loss to increase if k is increased? Briefly justify your answer.

    Answer: No. Using SVD to perform PCA, we are finding the components and sorting them based on their eigenvalues. We are then selecting the top k components. This means that increasing k does not affect the previous components, causing the loss to either decrease or stay the same.

11. Why doesn't it make sense to do PCA with $k > d$?

    Answer: In PCA, we are projecting the features onto a lower dimensional (ideally) space. Projecting the features into a higher dimensional space does not provide any benefits as the features will still live on the original manifold (albeit in a higher dimensional space), increasing space complexity without providing any benefits.

12. In terms of the matrices associated with PCA (X, W, Z, $\hat{X}$), where would an "eigenface" be stored?

    Answer: Each eigenface will be one row of W.

CPSC 340 Assignment 6 (due Wednesday December 7 at 11:55pm)

Name(s) and Student ID(s):

Answer: Masoud Mokhtari, 14186167 (Using 1 late day)

1 Sparse Latent-Factor Models

If run the script *example_faces.jl* it will load a set of face images under different lighting conditions. It will then fit a PCA model (with $k = 16$) and creates 3 plots:

- Random faces taken from the dataset (*facesOriginal*).
- Reconstructed versions of these faces based on the PCA model (*facesCompressed*).
- The average face (*faceMu*).
- The principal compoenents (*eigenfaces*).

Since Plots.jl seems to only allow you to show one plot at at time, I think you can type 'facesOriginal" in the REPL to switch to that plot (for example). The reconstructions tend to be reasonable given that they compress each 1024-pixel face down to just 10 numbers, although they often lack specific details and are less accurate for faces that do not look like the average face. In this model, some of the principal components are interpretable (for example, they seem to reflect different lighting conditions), but many of them are not. You can change the value of k in this script to see the effect of including/excluding principal components, if you increase k it will do a better job of reconstructing faces that do not look like the average face.

1.1 Uniqueness of Principal Components

On different runs of the script, you may get different principal components, even though all that changes between runs is the order of the training examples. What is the specific difference between the principal components that are obtained between different runs of the algorithm?

Answer: We can see that from time to time, the only thing that changes is the sign of the PCA components. Meaning that for some of them, the negative pixels become positive, and positive pixels become negative. That is sometime a PCA components gets multiplied by -1. To illustrate this, I provide one example in Figure 1, which illustrated the non-uniqueness of PCA.

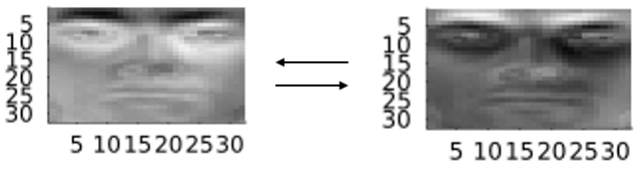


Figure 1: Showing one PCA components changing between runs.

1.2 Non-Negative Matrix Factorization

If you replace the function call to *PCA* with *PCA_gradient*, then instead of using the SVD to compute the principal components it will compute them numerically by using gradient descent steps (alternating between updating W and Z). Note that this returns different principal components because it does not enforce any constraints on W, but it will give an equivalent predictions (up to numerical accuracies).

A disadvantage of PCA is that the principal components tend to be dense (they are all non-zero), which can make them more difficult to interpret. One of the first sparse variations on PCA is non-negative matrix factorization, where we use the PCA objective but add non-negative constraints on W. Using *PCA_gradient* as a template, write a function *NMF* that implements the non-negative matrix factorization (NMF) model.

Hint: you need to make several changes. First, you need to remove the step that centers the data matrix X (NMF cannot model the negative values that get introduced by centering). Second, you need to change the initialization so that negative values of W and Z are set to 0. Third, you need to change the updates of W and Z to use an optimization method that includes the non-negativity constraint. You can use the function *findMinNN* which minimizes a differentiable function subject to non-negative constraints. You will also need to update the *compress* function so that it does not center the data and it finds the non-negative Z minimizing the objective with W fixed, and the *expand* function so that it does not "un-center" the data.

1. Hand in the 3 updated functions required to implement NMF instead of PCA.

   Answer: The implementations are shown in Figures 2, 3, and 4.

```
function PCA_gradient_NMF(X,k)
    (n,d) = size(X)

    # Initialize W and Z
    W = randn(k,d)
    Z = randn(n,k)

    # Set negative values to 0
    W[W.<0] .= 0
    Z[Z.<0] .= 0

    R = Z*W - X
    f = sum(R.^2)
    funObjZ(z) = pcaObjZ(z,X,W)
    funObjW(w) = pcaObjW(w,X,Z)
    for iter in 1:50
        fOld = f

        # Update Z
        Z[:] = findMinNN(funObjZ,Z[:],verbose=false,maxIter=10)

        # Update W
        W[:] = findMinNN(funObjW,W[:],verbose=false,maxIter=10)

        R = Z*W - X
        f = (1/2)sum(R.^2)
        @printf("Iteration %d, loss = %f\n",iter,f/length(X))

        if (fOld - f)/length(X) < 1e-2
            break
        end
    end
```

Figure 2: Implementation of the NMF variant of PCA$_g radient$.

```
function compress_gradientDescent_NMF(Xhat,W)
    (t,d) = size(Xhat)
    k = size(W,1)
    Z = zeros(t,k)

    funObj(z) = pcaObjZ(z,Xhat,W)
    Z[:] = findMinNN(funObj,Z[:],verbose=false)
    return Z
end
```

Figure 3: Implementation of the NMF variant of the compress function.

```
function expandFunc_NMF(Z,W)
    (t,k) = size(Z)
    return Z*W
end
```

Figure 4: Implementation of the NMF variant of the expand function.

2. What do you think the NMF factor represents if you set $k = 1$? (Hint: look at all the plots made by the original script.)

   Answer: Running the original script and looking at the "faceMu" plot, we see the image shown in Figure 5 (Left). Setting $k = 1$ and running the NMF variant, for the PCA component, we get the image shown in Figure 5 (Right). This means that the NMF with $k = 1$ is choosing the non-negative average face as its PCA component. The reason for this happening could be that we don't center the

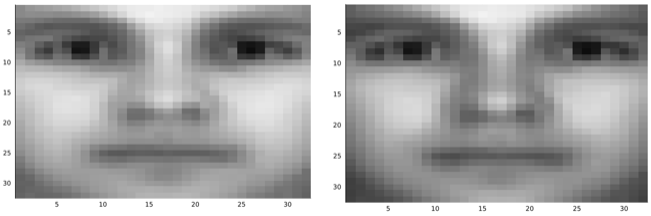


Figure 5: (Left) faceMu of model run by the original script. (Right) PC of the NMF variant with $k = 1$.

   data when using NMF, and when $k = 1$, this average face becomes an important PC. We also know that when $k = 1$, we find the direction that minimizes the squared orthogonal distance.

3. What do the NMF factors represent if you set $k = 2$?

   Answer: With $k = 2$, as shown in Figure 6, the components represent two different lighting conditions where either the left or right side of the face is brighter.

4. Hand in a plot of the NMF factors with $k = 16$.

   Answer: The components are shown in Figure 7.

5. Explain why the NMF model should have a higher or lower loss than the PCA model.

   Answer: Using NMF is similar to adding regularization to our PCA, and in regularized cases, we expect E_{train} to increase while E_{approx} decreases. Therefore, in this case, we expect the loss printed by the model (which is the loss on the reconstruction of the training data) **to be higher**. We do, however, expect the model to have lower loss for samples that are different from the average face.

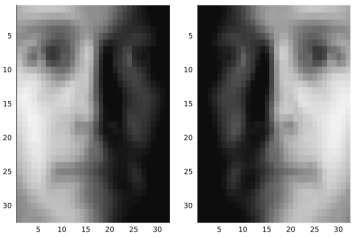


Figure 6: Principle components of the NMF model with $k = 2$.

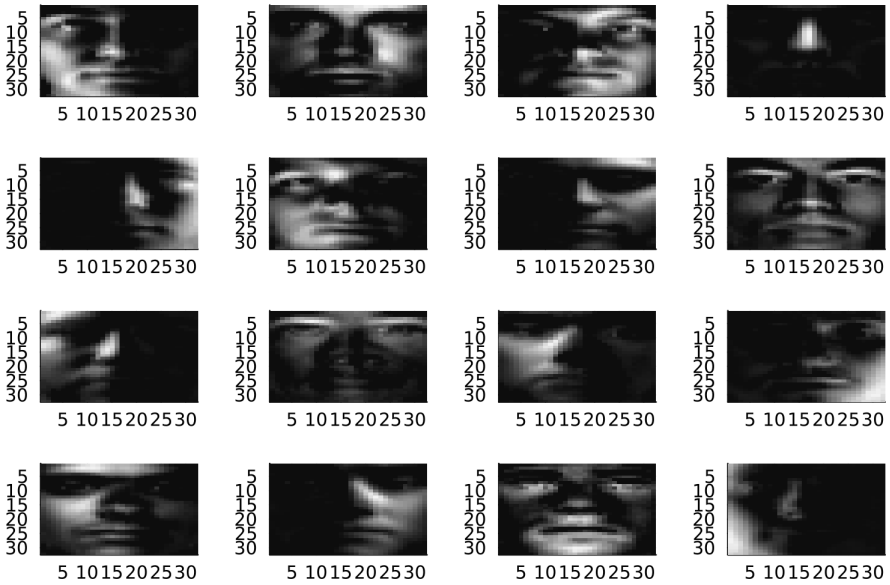


Figure 7: Principle components of the NMF model with $k = 16$.

2 Neural Networks

2.1 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 $(x^i)^T = \begin{bmatrix} -3 & -2 & 2 \end{bmatrix}$ 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: A single-layer NN for a single sample takes the form:

$$\hat{y}_i = v^T h(W x_i)$$

Therefore:

$$z^i = \begin{bmatrix} -2 & 2 & -1 \\ 1 & -2 & 0 \end{bmatrix} \begin{bmatrix} -3 \\ -2 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$a^i = \sigma(z^i) = \begin{bmatrix} \frac{1}{1+e^0} \\ \frac{1}{1+e^{-1}} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{1+e^{-1}} \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.731 \end{bmatrix}$$

$$\hat{y^i} = \begin{bmatrix} 3 & 1 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.731 \end{bmatrix} = 2.231$$

2.2 Neural Network Tuning - Regression

The file `example_nnet.jl` runs a stochastic gradient method to train a neural network on the *basisData* dataset, and outputs a plot of the result (depending on your configuration, it may output these plots as the algorithm is running too so that can see how the learning proceeds). Unfortunately, in its current form the neural network does not fit the data 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 feautres, 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: The final modified plot is shown in Figure 8.

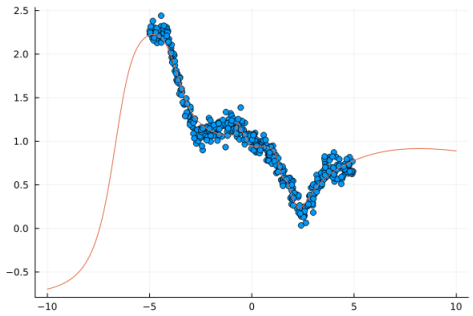


Figure 8: Final modified NN plot for regression.

The original plot with no changes is shown in Figure 9, and the following changes were made to improve performance (we show incremental plots as well to see the effect of each change).

- Added bias by modifying the input matrix during both training and inference and also increased the number of iterations to 10^5. We get the plot shown in Figure 10.
- Added support for mini-batch optimization (with a batch size of 200). This change was not trivial and required some changes to the "NeuralNet_backprop" function as the matrix dimensions did not originally match. Additionally, the loss function had to be changed (forward: $f = (1/n) * sum(r.^2)$, backprop: $dr = 2/n. * r$) to account for the size of the minibatch. The improved plot after this change is shown in Figure 11.
- Added more hidden layers by modifying nHidden to $nHidden = [15, 5, 3]$. The improved plot after this change is shown in Figure 12.
- Lastly, instead of modifying the step size, we simply use findMin with our new mini batch-enabled "NeuralNet_backprop" function, which dynamically modifies the step size. The resuling plot is the one shown as the final plot in Figure 8.

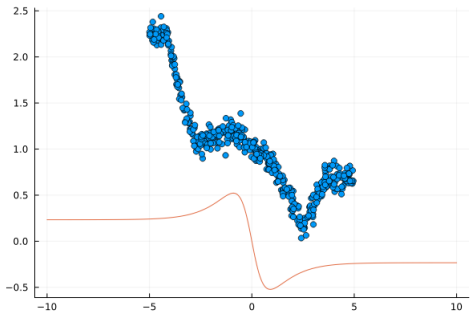


Figure 9: NN plot with no changes.

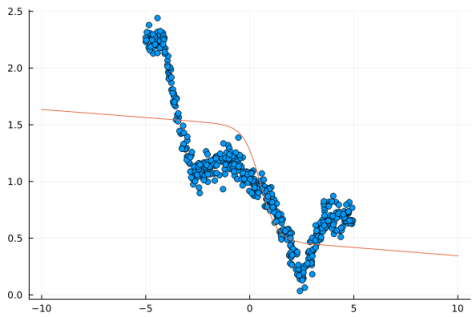


Figure 10: NN plot with added bias and more training iterations.

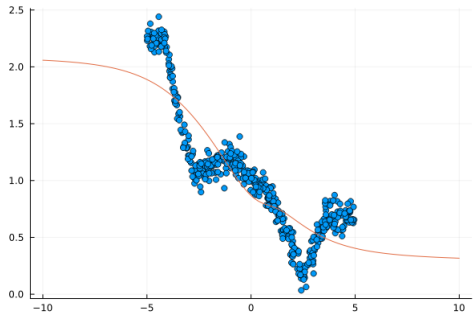


Figure 11: Added mini batch support.

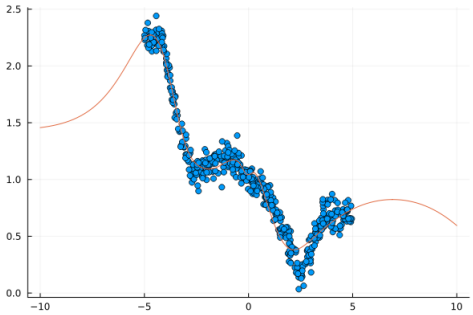


Figure 12: Added more hidden layers.

2.3 Neural Network Tuning - Classification

The file `example_mnist35_logistic` runs a stochastic gradient method to train a neural network on the '3' and ''5' images from the MNIST dataset, which obtains a test error of 0.03. If you instead run `example_mnist35_nnet.jl`, it fits a neural network with stochastic gradient descent on the same data (using the squared error for training and classifies by taking the sign of the prediction) but obtains a worse error. Modify the neural network 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 13, we achieve an error of 0.01 by adding the following changes:

- Added support for mini batches and use a mini batch size of 300.
- Standardized each image by finding the mean and std of the training data and using $X = X. - mean./std.$
- Added a bias term to both training and test samples by adding a column of 1's to the feature matrices.
- We add an output Sigmoid activation function and use Cross Binary Entropy for the loss rather than using a regression loss.
- We change the number of layers by setting $nHidden = [20, 15, 5]$
- Lastly, we add a reduce-on-plateau scheduler where the learning rate is reduced with a patience of 3 (error doesn't decrease for 3 iterations). We start with an initial step size of 10^-4.

```
julia> include("example_mnist35_nnet_mod.jl")
Training iteration = 0, error rate = 0.37
Training iteration = 1000, error rate = 0.03
Training iteration = 2000, error rate = 0.02
Training iteration = 3000, error rate = 0.02
Training iteration = 4000, error rate = 0.02
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
Training iteration = 11000, error rate = 0.02
Training iteration = 12000, error rate = 0.01
```

Figure 13: Our changes allow us to achieve an error of 0.01.

3 Using a Deep Learning Package

The script *example_flux.jl* shows how the 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 question, as well 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. In 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 have a matrix W and a vector v?

   Answer: One advantage is that using advanced initialization methods such as Xavier or He is much easier when W and v are separately defined. Additionally, coding is much simpler with this approach since reshaping is not required, and it is easier to keep track of each matrix's dimension. Lastly, the content of W and v can be separately analyzed this way, which helps tremendously when debugging model's behaviour.

2. If we use 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 see two sets of parameters included in the Dense(W) layer. The first set of parameters have the shape (3, 784), and the second set has the shape (3,). The first set is the set of weights for our hidden layer that gets multiplied by the input features matrix. The second set oontains the bias terms, one for each hidden unit. This makes the overall number of parameters $3 \times 784 + 3 = 2355$.

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

   Answer: The code for the two hidden-layer implementation is shown in Figure 14.

```
# Randomly initialize the weights for W1 and W2 and v_transposed
W1 = randn(128,d)
W2 = randn(3, 128)
vt = reshape(v,1,3)

# Chain the two hidden-layer network
dense_model = Chain(Dense(W1, true, relu), Dense(W2, true, relu), Dense(vt, false, identity))

# Define the loss function
lossfunc(x,y) = (1/2)*(dense_model(x)[1]-y)^2

# Forward and backprop layers
f_layer = lossfunc(X[i,:],y[i])
g_layer = gradient(params(dense_model)) do
    lossfunc(X[i,:],y[i])
end
gW1_layer = g_layer[W1]
gW2_layer = g_layer[W2]
gv_layer = g_layer[vt]
```

Figure 14: 2 Hidden layers using Flux's Chain module.

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

   Answer: Firstly, we have 6 filters in this network. Each filter is associated with one bias term. That is 6 parameters for the bias terms. Secondly, each filter has $5 \times 5 \times 3 = 75$ parameters, for a total of $6 \times 75 = 450$. Therefore, with the bias terms, we get $450 + 6 = 456$ parameters.

5. Re-write the model and training procedure you developed for Question 2.3 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 runtime, memory, or accuracy).

   Answer: First, we list the changes we made for Question 2.3, and for each change, we state how Flux's implementation differ:

   - Added support for mini batches and use a mini batch size of 300 - Same implementation for Flux.
   - Standardized each image by finding the mean and std of the training data and using $X = X. - mean./std$ - Same implementation for Flux.
   - Added a bias term to both training and test samples by adding a column of 1's to the feature matrices - No need to manually do this for Flux, the Dense layers take care of this.
   - We add an output Sigmoid activation function and use Cross Binary Entropy for the loss rather than using a regression loss - We add the Sigmoid function by using the proper parameter of the Dense function and use Flux's binarycrossentropy loss instead of manually writing this.
   - We change the number of layers by setting $nHidden = [20, 15, 5]$ - This is reflected into how the Dense layers are added inside the Chain module.
   - Lastly, we add a reduce-on-plateau scheduler where the learning rate is reduced with a patience of 3 (error doesn't decrease for 3 iterations). We start with an initial step size of 10^-4. - Same implementation for Flux.

   We provide the code in Figure 15. And below, I explain the performance differences I noticed:

   - Accuracy: As shown in Figure 16, this model achieves a similar error of 0.01 as our prior implementation.
   - Runtime: Flux's implementation provides a general solution where many different networks can be easily defined and automatically trained. This comes at the cost of higher runtime compared to our

```julia
# Load X and y variable
using JLD, Printf, Statistics
using Flux
using Flux.Optimise: update!
data = load("mnist35.jld")
(X,y,Xtest,ytest) = (data["X"],data["y"],data["Xtest"],data["ytest"])
y[y.==2] .= 0
ytest[ytest.==2] .= 0
(n,d) = size(X)

feats_std = std(X)
feats_mean = mean(X)

# Standardize data
X = X.-feats_mean ./ feats_std
Xtest = Xtest.-feats_mean ./ feats_std

# Define a 3 hidden layer model using Flux
model = Chain(Dense(d, 20, tanh),
              Dense(20, 15, tan),
              Dense(15, 5, tanh),
              Dense(5, 1, sigmoid))

# Define the loss function
loss(x, y) = Flux.binarycrossentropy(model(x), y)

# Extract model params to be used for grad computation
ps = params(model)

# Define the optimizer
stepSize = 1e-2
opt = Descent(stepSize)

# Train with mini batches
maxIter = 50000
samples_per_batch = 300
patience = 3
prev_err = 9999
for t in 1:maxIter

    i = rand(1:n, samples_per_batch)

    grads = Flux.gradient(ps) do
        loss(transpose(X[i, :]), reshape(y[i], 1, size(y[i], 1)))
    end
    update!(opt, ps, grads)

    # Every few iterations, plot the data/model:
    if (mod(t-1,round(1000)) == 0)
        yhat = model(transpose(Xtest)) .> 0.5
        err = sum(yhat .!= reshape(ytest, 1, size(ytest, 1)))/size(Xtest,1)
        @printf("Training iteration = %d, error rate = %.2f\n",t-1,err)

        if err < prev_err
            global patience = 3
            global prev_err = err
        else
            if patience == 1
                global stepSize /= 10
                global opt = Descent(stepSize)
                @printf("Reduced step size to %.4f", stepSize)
                global patience = 3
            else
                global patience -= 1
            end
        end
    end
end
```

Figure 15: Flux implementation of Q2.3.

implementation for Q2.3. This is due to Flux's auto-differentation having the same computatoinal complexity as the forward path of the network.

- Memory: Flux's implementation has higher memory consumption because its auto-differentiation operations need to save some intermediate results during forward propagation to be used during gradient computation.

```
Training iteration = 0, error rate = 0.64
Training iteration = 1000, error rate = 0.04
Training iteration = 2000, error rate = 0.03
Training iteration = 3000, error rate = 0.03
Training iteration = 4000, error rate = 0.02
Training iteration = 5000, error rate = 0.02
Training iteration = 6000, error rate = 0.01
```

Figure 16: Error attained by the Flux implementation.

4 Very-Short Answer Questions

1. Consider fitting a linear latent-factor model, using L1-regularization of the w_c values and L2-regularization of the z_i values,

$$f(Z,W) = \frac{1}{2}\|ZW - X\|_F^2 + \lambda_W \sum_{c=1}^{k} [\|w_c\|_1] + \frac{\lambda_Z}{2} \sum_{i=1}^{n} \left[\|z_i\|^2\right],$$

   (a) What is the effect of λ_Z on the two parts of the fundamental trade-off in machine learning?

   Answer: Increasing λ_Z increases the regularization effect, which increases E_{train} and decreases E_{approx}.

   (b) What is the effect of k on the two parts?

   Answer: Increasing k increases the number of principle components used, which decreases E_{train} and increases E_{approx}.

   (c) Would either of answers to the *previous two questions* change if $\lambda_W = 0$?

   Answer: If we set $\lambda_W = 0$, we are only enforcing regularization on z, which is a wrong approach. This way, decreasing values of z can cause the values of w to increase and vice versa. Therefore, changing λ_Z has no effect, while changing k behaves the same as part b.

2. Which is better for recommending movies to a new user, collaborative filtering or content-based filtering? Briefly justify your answer.

   Answer: Content-based filtering. This is because collaborative filtering relies on the similarity between user rows and movie columns to make recommendations (Using latent factors), and a new user will have an empty, non-informative row in this matrix. Conversely, content-based filtering uses the features of the new user to make a recommendation.

3. 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: In total, there are $1000 + 100 + 300 + 3 = 1403$ parameters. Here is the breakdown: The matrix transforming the input features to the hidden dimension contains $10 \times 100 = 1000$ parameters plus 100 bias params. The hidden to output projection matrix has $100 \times 3 = 300$ parameters plus 3 bias terms.

4. Consider fitting a neural network with one hidden layer. Would we call this a parameteric 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 first case, the memory is not independent of input samples, therefore, the **model is non-parameteric**. In the second case, the memory complexity is O(1) with regards to n, therefore, **the model is parameteric.**

5. 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: This can be achieved by first over-fitting the model to the original set of features and finding a minimum. We can then generate new relevant features and continue training the model with these added features. This way, the model stays within the same minimum, but further explores the space to decrease test error by finding a more regularized solution.

6. 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: For a single-sample input, the final answer is $O(kd + (L-1)k^2 + ck)$. Here is the breakdown: For the initial layer, we have $O(kd)$ for the matrix multiplication. For the consecutive layers (except for the last), we have $O(k^2)$, so $(L-1)k^2$ in total. Lastly, the last layer has O(ck).

7. What is the cost of backpropagation in the previous question?

   Answer: $O(kd + (L-1)k^2 + ck)$ since for gradient computation (in the special case of AD that we use), forward and backprop have the same computational complexity.

8. What is the "vanishing gradient" problem with neural networks based on sigmoid non-linearities?

   Answer: At the two tails of the Sigmoid non-linearity, the function plateaus, which causes the gradient at these points ot be 0. This is even worsened for nested Sigmoids. Such 0 gradient values cause the model to stop learning and updating its weights, which is known as the vanishing gradient problem.

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

   Answer: One advantage is that a wide range of forward functions can be used without having to explicitly find the formulation for gradients, which is difficult and error-prone. One disadvantage is that the forward and backprop complexity is the same, whereas, for hand-computed gradients, the backprop can be much faster. Additionally, auto-differentation (based on variant) can require extra memory to save intermediate values.

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

    Answer: The number of parameters can be intractably high if we do image classification with fully connected neural networks. For example, for a $224 \times 224 \times 3$ image, the flattened vector has a dimension of 150,528, causing a large number of parameters to be needed. In contrast, the number of parameters for convolutions does not depend on image size and can be controlled via selected filter dimensions. Additionally, with fully connected NNs, we are losing important spatial information by flattening the image.