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):
Sam Coble, 23650526
Foo Chuan Shao, 30321277

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?

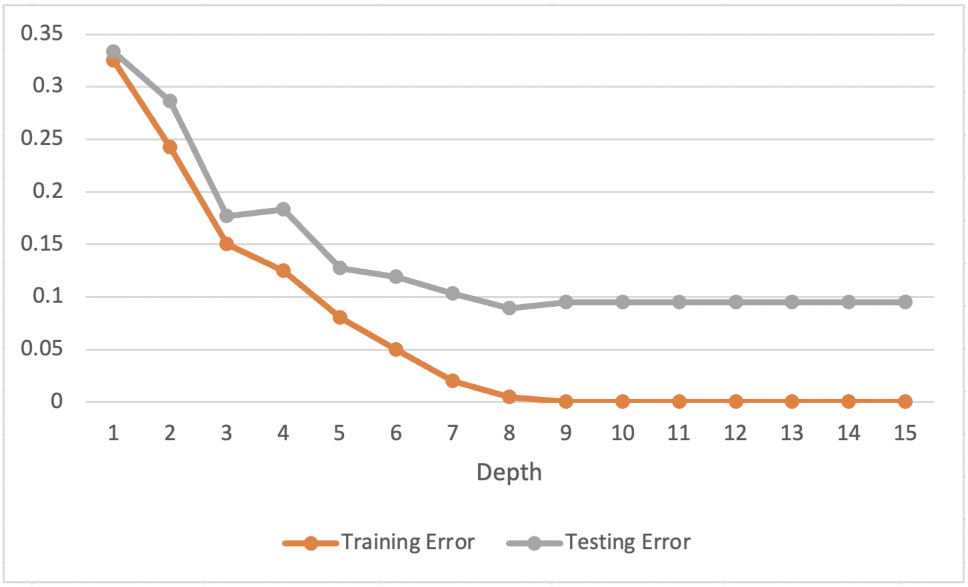
Answer: The training error with accuracy-based tree starts at a lower value, but eventually reaches a minimum error of 0.11. Whereas, the infogain-based decision tree starts at a high infogain value and eventually reaches 0 additional info gain.
An explaination for training error might be that with depth, the training error decreases as expected as the model becomes more detailed and specific. However, it reaches a peak as there are outliers that the decision tree is unable to fit to.
An explaination for infogain-based decision tree might be that in the first depth, there could be much information for it to learn as it is all new. However, as the depth increases, there is less new information for it to learn, and eventually hits 0 when it has been fitted and there is no more information to learn.

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 training error of the model decreases as the depth decreases to reach nearly 0. Whereas, the test error decreases but has a point where it stops decreasing at a certain depth.

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: We would use a depth of 3, which minimizes the validation error of the second half of the training error to 0.073. If we swap the training and validation sets, so that the first half of the data is used for validation and the second half is used for training, we get the best validation error of 0.060 with a model of depth equal to 5. We could use more of our data the estimate the depth by doing cross-validation, where the validation set becomes different part of the total available data for multiple different runs and then the most common chosen hyper-parameter value (the depth) is selected for the final model.

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

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

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

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: $1/7$

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

  Answer: $1/7$

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

  Answer: $4/7$

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

  Answer: 4/5

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

  Answer: 3/5

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

  Answer: 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: Assume independence of features
$p(\text{spam} = 1 \mid \text{<your name >} = 1, \text{pharmaceutical} = 0, \text{PayPal} = 1) = p(\text{<your name>} = 1 \mid \text{spam}) * p(\text{pharmaceutical} = 0 \mid \text{spam}) * p(\text{PayPal} = 1 \mid \text{spam})$
$= 1/7 * 1/7 * 4/7 = 4/343$

$p(\text{Not spam} = 1 \mid \text{<your name >} = 1, \text{pharmaceutical} = 0, \text{PayPal} = 1) = 4/5 * 3/5 * 1/5 = 12/125$

The most likely label for the test example is "Not Spam" as the probability of being "Spam" is much smaller than the probability of the message being "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: 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: rec.*

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.

```
p_xy = zeros(2,d,k)
labelCounts = zeros(k)
for i in 1:n
  # count total occurances of each label
  labelCounts[y[i]] = labelCounts[y[i]] + 1
  for j in 1:d
    # count how many times a feature appeared given a label
    if X[i,j] == 1
      p_xy[1,j,y[i]] = p_xy[1,j,y[i]] + 1
    else # count how many times a feature didn't appear given a label
      p_xy[2,j,y[i]] = p_xy[2,j,y[i]] + 1
    end
  end
end

# percentage of time a label appeared/didn't appear given a feature
for j in 1:d
  for c in 1:k
    p_xy[1,j,c] = p_xy[1,j,c] / labelCounts[c]
    p_xy[2,j,c] = p_xy[2,j,c] / labelCounts[c]
  end
end
```

Answer: Test error with naive Bayes: 0.188

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: $O(tdk)$

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.

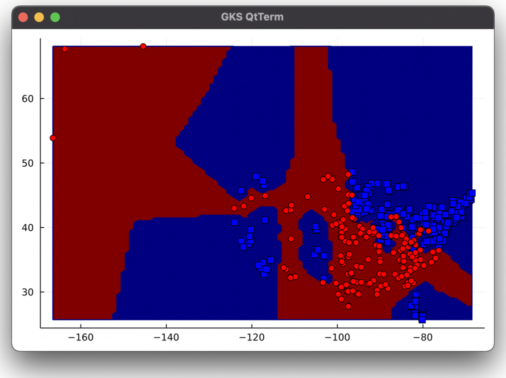
1. Hand in the predict function.

   Answer: On next page

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, Testing Error=0.065
   k=2: Training Error=0.035, Testing Error=0.092
   k=3: Training Error=0.028, Testing Error=0.066

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



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: BLUE

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

   Answer: As the nearest neighbour to any point in the same dataset is itself, by KNN's algorithm, its predicted result would also be itself. Therefore, the predicted result on the training dataset will always correct and cause the training error to be 0

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

   Answer: Square-root of total examples so far

```
function knn_predict(Xhat,X,y,k)
  (n,d) = size(X)
  (t,d) = size(Xhat)
  k = min(n,k) # To save you some debuggin
  yhat = zeros(t)
  for i in 1:t
    kNearest = fill(Inf, 2, k)
    for j in 1:n
      distToJ = sqrt(sum((X[j,:]-Xhat[i,:]).^2))
      maxDist = maximum(kNearest[1,:])
      if distToJ < maxDist
        m = findall(x -> x == maxDist, kNearest[1,:])[1]
        kNearest[1,m] = distToJ
        kNearest[2,m] = y[j]
      end
    end
    yhat[i] = mode(kNearest[2,:])
  end
  return yhat
end
```

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: Splitting of the decision tree always stops when only one example is left in a leaf.

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

   Answer: This is as the random tree only considers d randomly-chosen features. Therefore, even at a depth of infinity, it will not fully train as the tree will overlook some features, resulting in a training error greater than 0.

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: Code on next page.

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: Training Error: 0.000, Testing Error: 0.167. This is about half of the testing error of a single info-gain decision tree, and is way better than a single random tree. This is about what I expected as the training error goes back to zero like the single info-gain decision tree because all features and examples become represented, and the testing error reduces relative to the info-gain method as there is less chance of overfitting.

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

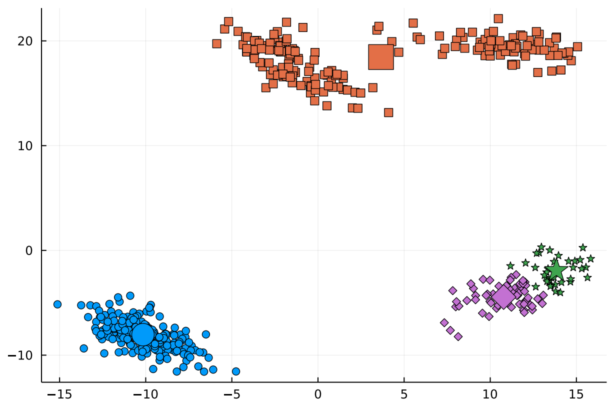
   Answer: With enough random trees, the likelihood that an important feature is not represented or is underrepresented approaches zero, and therefore every example and every feature is counted a sufficient number of times to make zero mistakes on the training data.

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

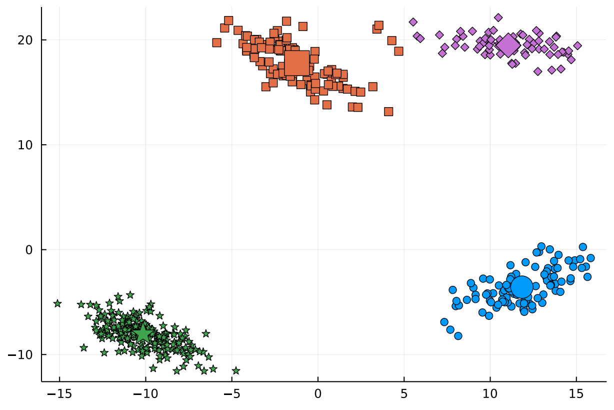
```
function randomForest(X, y, depth, nTrees)
        subModels = Array{GenericModel}(undef, nTrees)
        for i in 1:nTrees
                subModels[i] = randomTree(X, y, depth)
        end
        return subModels
end

function predictForest(subModels, Xtest)
        (n, d) = size(Xtest)
        nTrees = size(subModels)[1]
        predictions = fill(0.0, (nTrees, n))
        for i in 1:nTrees
                predictions[i,:] = subModels[i].predict(Xtest)
        end
        for i in 1:n
                yhat[i] = mode(predictions[:,i])
        end
        return yhat
end
depth = Inf
nTrees = 50
subModels = randomForest(X, y, depth, nTrees)
yhat = predictForest(subModels, X)
trainError = mean(yhat .!= y)
@printf("Train Error with depth-%d random forest with n-%d:
↪  %.3f\n",depth,nTrees,trainError)
yhat = predictForest(subModels, Xtest)
testError = mean(yhat .!= ytest)
@printf("Test Error with depth-%d random forest with n-%d:
↪  %.3f\n",depth,nTrees,testError)
```

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},

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

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.

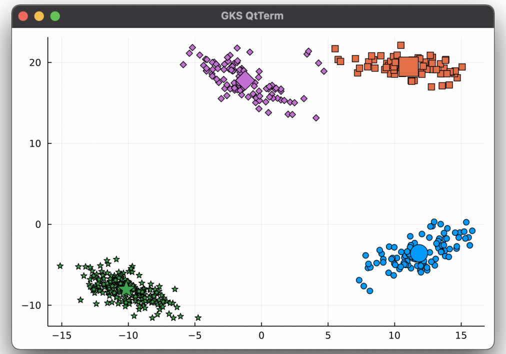
   Answer: Code below.

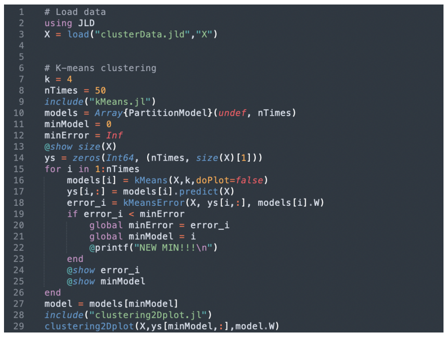
2. Instead of printing the number of labels that change on each iteration, what trend do you observe if you print the value of *kMeansError* after each iteration of the k-means algorithm?

   Answer: The error begins in the mid-5-digit range, before eventually converging to either around 3000 if the 'correct' labels are chosen, or around 9000 if the 'incorrect' labels are chosen.

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:





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


# K-means clustering
k = 4
nTimes = 50
include("kMeans.jl")
models = Array{PartitionModel}(undef, nTimes)
minModel = 0
minError = Inf
@show size(X)
ys = zeros(Int64, (nTimes, size(X)[1]))
for i in 1:nTimes
    models[i] = kMeans(X,k,doPlot=false)
    ys[i,:] = models[i].predict(X)
    error_i = kMeansError(X, ys[i,:], models[i].W)
    if error_i < minError
        global minError = error_i
        global minModel = i
        @printf("NEW MIN!!!\n")
    end
    @show error_i
    @show minModel
end
model = models[minModel]
include("clustering2Dplot.jl")
clustering2Dplot(X,ys[minModel,:],model.W)
```

```
function distanceSquared(a, b)
        d = size(a)[1]
        sum = 0
        for i in 1:d
                sum = sum + (a[i] - b[i]) * (a[i] - b[i])
        end
        return sum
end
function kMeansError(X, y, W)
        (n, d1) = size(X)
        (k, d2) = size(W)
        if d1 != d2
                @printf("non-matching dimensions of X and W\n")
        end
        error = 0
        for i in 1:n
                error = error + distanceSquared(X[i,:], W[y[i],:])
        end
        return error
end
```

5.2 Selecting k in k-means

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

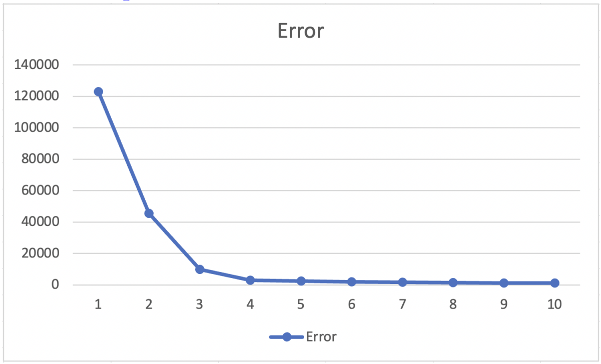
1. Explain why the *kMeansError* function should not be used to choose k.

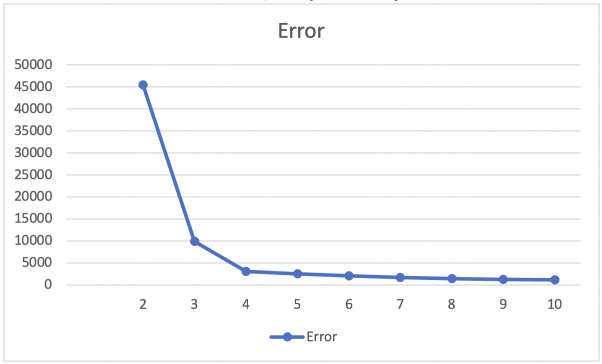
   Answer: The error will always decrease when k increases, until k goes to n, as it is based on the sum (approximately n/k per mean) of squares of all distances to each respective mean (which is itself when k=n), so error will be 0 at k=n.

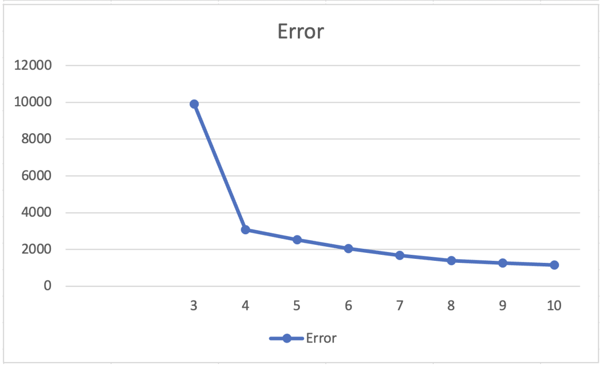
2. Explain why even evaluating the *kMeansError* function on test data still wouldn't be a suitable approach to choosing k.

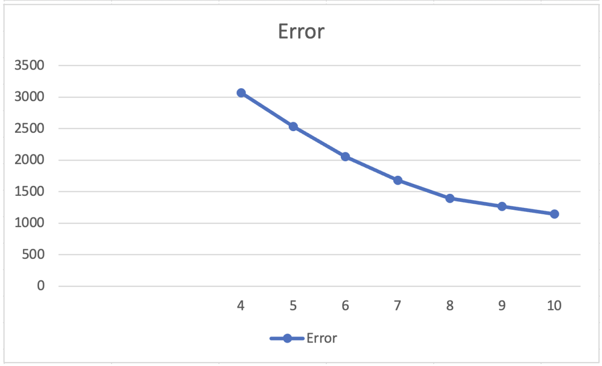
   Answer: The higher k is, the amount of data points per mean will decrease on average, therefore also decreasing the sum. Therefore, k will tend to n.

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









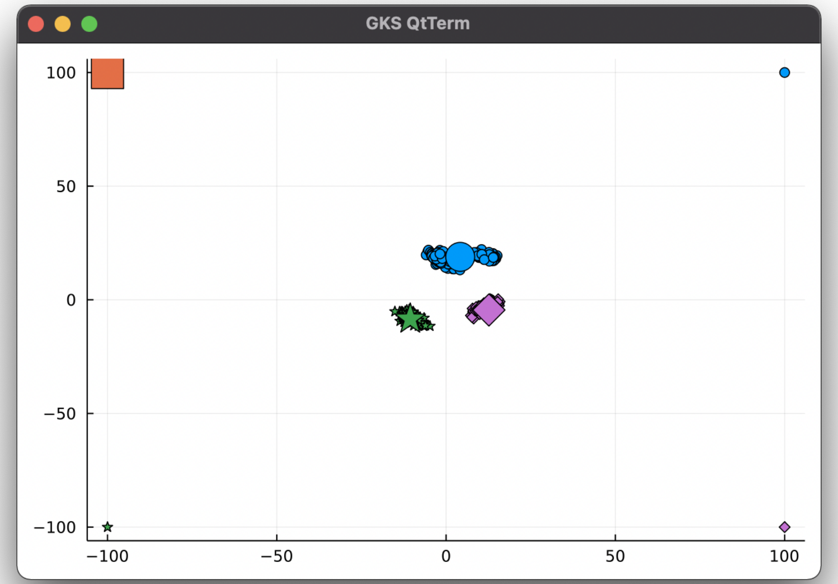
4. The *elbow method* for choosing k consists of looking at the above plot and visually trying to choose the k that makes the sharpest "elbow" (the biggest change in slope). What values of k might be reasonable according to this method? Note: there is not a single correct answer here; it is somewhat open to interpretation and there is a range of reasonable answers.

   Answer: K=4 seems the most reasonable for this method, as after that, the decrease in error is roughly linear, however 4 is still a very large decrease in error from k=3. K=3 could also work, however, the drop off to four is less optimal when looking for the "elbow shape", as the curve from vertical to horizontal is much smoother.

5.3 k-Medians

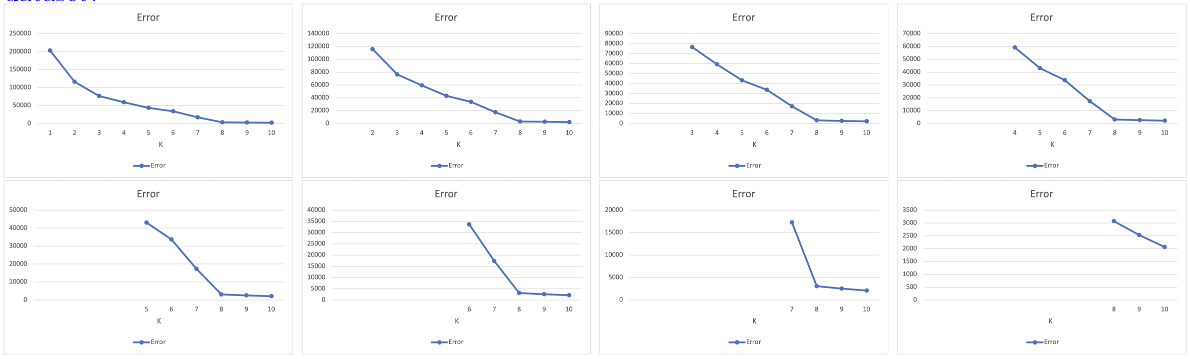
The data in *clusterData2.mat* is the exact same as the above data, except it has 4 outliers that are very far away from the data.

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: This is not a satisfying value for k because the outliers should not in the same clusters as the data that is in the center of the chart. Additionally, there is a cluster made of up just one example in the corner.

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



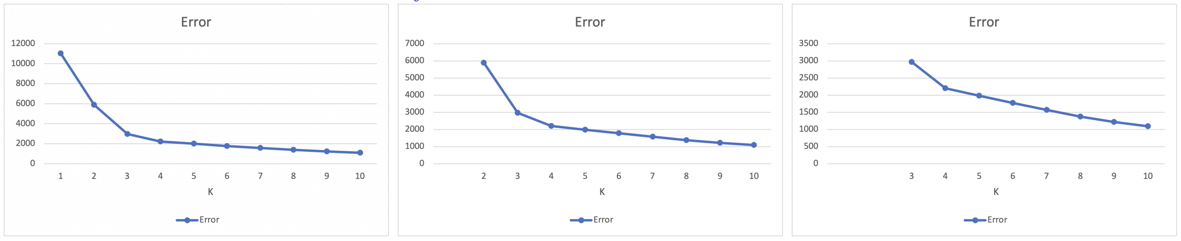
   Answer: A value of K=8 would be the most likely to be chosen for this dataset using the elbow method, as there is a sever change in the slope of the elbow plot at this value.

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,

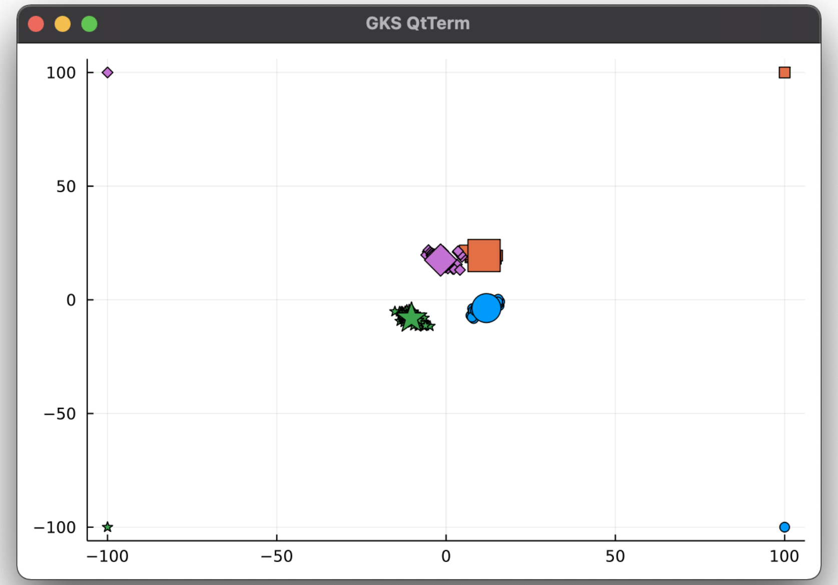
$$f(w_1, w_2, \ldots, w_k, y_1, y_2, \ldots, y_n) = \sum_{i=1}^{n} \|x_i - w_{y_i}\|_1 = \sum_{i=1}^{n} \sum_{j=1}^{d} |x_{ij} - w_{y_i j}|.$$

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: I would choose a value of k=4 when using the L1-norm, however, a value- of k=3 also looks good using the elbow method.

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: Code below:

```
function L1Dist(a, b)
        d = size(a)[1]
        sum = 0
        for i in 1:d
                sum = sum + abs(a[i] - b[i])
        end
        return sum
end
function L1Distances(A, B)
        (n, d1) = size(A)
        (t, d2) = size(B)
        ret = zeros(Float64, n, t)
        for i in 1:n
                for j in 1:t
                        ret[i,j] = L1Dist(A[i,:], B[j,:])
                end
        end
        return ret
end
function kMediansError(X, y, W)
        (n, d1) = size(X)
        (k, d2) = size(W)
        if d1 != d2
                @show "non-matchin dimensions of X and W"
        end
        error = 0
        for i in 1:n
                error = error + L1Dist(X[i,:], W[Int(y[i]),:])
        end
        return error
end

function kMedians(X,k;doPlot=false)
        # K-medians clustering

        (n,d) = size(X)

        # 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 median
                D = L1Distances(X, W)

                # 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

                # @printf("Running k-means, changes = %d, error = %f\n",changes, kMeansError(X, y, W))
            end

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

                D = distancesSquared(Xhat,W)
                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
```

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: Some feature transformation that can be applied would be feature aggregation, discretization, or feature selection. This will reduce the number of features such that there would be less "coupons" to collect, making it faster to learn.

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

   Answer: One reason could be to understand the relationship of the data or to identify outliers in the dataset for preprocessing.

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

   Answer: They would all result in the same model as doing $\leq$ is equvalent to doing $<$ on the next highest feature, and doing $>$ will just switch the labels, before they are switched back again.

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

   Answer: Certain words are more likely to be found in the presense of other words. For example, in an email containing the word "cpsc", the word "340" is more likely to appear.

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

   Answer: A validation a part of the training set that is used to simulate the test set to find optimal hyper-parameters after training multiple models, as you must never let the test set influence training in any way.

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

   Answer: In the example for a decision tree, maximizing the depth will increase the training error, but overfit the model. In the general case, using training error to select hyper-parameter would cause the model to be overfitted and not being able to generalize to new data.

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: 10*16*4320/24/60/60 = 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: It is necessary to find the probabilities without having an incredibly insane amount of data. Without it, it would be physically impossible to determine the probability of a label given a specific combination of features, as the probability of that feature combination would only be able to be known with examples of that exact feature combination, i.e., the exact same input.

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

   Answer: KNN is non-parametric because the model grows as more data is given. Disadvantages of non-parametric models can be the high cost to store the possibly theoretically infinite model size and the time it takes to classify an example.

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

Answer: By increasing the number of training examples, in general, data would increase the training accuracy, but decrease the approximation accuracy. However, as parametric models are simple, at a certain point, more data may not be necessarily useful to train any further.

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: Use data augmentation to increase the amount of data by making small translations in volume, pitch, time, etc. to every piece of original data.

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: Unsupervised learning (clustering) does not require labels to to the data.

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, they are not guaranteed to be convex