CPSC 440/540 Machine Learning (January-April, 2021) Assignment 1 (due January 22nd at midnight)

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

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

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

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

Basic Information

1. Name: Ricky Ma
2. Student ID: 82943424

1 Very-Short Answer Questions

1. A common pre-processing operation is to *standardize* our features. This operation replaces each x_j^i with $(x_j^i - \mu_j)/\sigma_j$, where μ_j is the mean of feature j on the training data and σ_j is the standard deviation of feature j on the training data. When doing predictions with the model, some codes standardize the test features $\tilde{x}_j^i$ with a mean $\tilde{\mu}_j$ and standard deviation $\tilde{\sigma}_j$ computing based on the test data. Describe why this usually is a mistake, and describe a setting where it can lead to terrible performance even if the training and test data come IID from the same distribution.
   The test data should remain pure, without any transformations done to it. In the real world, we computing the sample mean and sample s.d. can result in incorrect biases in our model since we do not know if the training data we used is an accurate representation of the test data.

2. What is the difference between a test set error and the test error?
   The test error is the "true" expected error computed over all test examples, while the test set is a sample used to approximate the true test error by computing the test set error.

3. In the deep learning world, *neural architecture search* describes the process of searching for the hyper-parameters of a neural network model (like the number of layers, the step size, whether to use convolutions, the regularization parameters, and so on) in order to optimize the performance on a fixed validation set. What is a potential problem with this approach?
   This process may lead to overfitting to the fixed validation set, and since the validation set influences training, this means we've violated the golden rule.

4. In a parametric model, what is the effect of the number of features d that our model uses on the training error and on the approximation error (the difference between the training error and test error)?
   In general, as d increases, training error decreases. Approximation error would first decrease, then likely increase due to overfitting.

5. Give a way to set the depth and width of a neural network that makes the model parametric, and a choice that makes the model non-parametric.
To make a parametric model, we can simply create a neural network with a fixed architecture (i.e. 2 hidden layers w/ 10 units each). To make the model non-parametric, we can make the depth and width of the neural network dependent on the sample size (i.e. hidden layers = N/100, units/layer = N/10).

6. You can fit a decision stump by finding the stump that maximizes the number of times $\hat{y}^i = y^i$ in the training data. Why do we not try to maximize this quantity when we fit the regression weights w in linear regression models?
$\hat{y}^i = y^i$ is used for finding classification error and does not apply to linear regression models. In regression, we want to know how "close" $\hat{y}^i$ is to y^i, for example, by finding $(\hat{y}^i - y^i)^2$

7. How does λ in an L1-regularizer affect the sparsity pattern of the solution (number of w_j set to exactly 0), the training error, and the approximation error?
As λ increases from 0, the number of w_j set to exactly 0 increases. In general, this prevents/reduces overfitting, meaning training error may increase, but approximation error will decrease (depending on the value of λ).

8. Minimizing the squared error used by in k-means clustering is NP-hard. What would be the significance of an algorithm that minimizes this error that runs in time $O(nd^2 + d^3)$.
This would mean that we have found a polynomial time algorithm that solves an NP-hard problem, which would give polynomial time solutions to all the problems in NP, proving that P=NP.

9. Consider the regression objective given by $f(w) = \sum_{i=1}^{n} \max\{0, |y^i - w^T x^i| - \epsilon\} + \frac{\lambda}{2}\|w\|^2$ for some number ϵ. Describe a method to minimize this objective function (or closely approximate the minimum).
We can use gradient descent by numerically approximating the gradient and taking steps in the opposite direction of the gradient to closely approximate the minimum. We can improve performance with various optimization techniques, like random initializations/restarts, adaptive learning rates, etc.

10. Consider an ensemble clustering method that generates m different bootstraps of the data. It then fits a k-means model (with a random initialization) to each of the bootstraps. To form the final clustering for example x_i, it chooses the y_i that is most common across the m clusterings. Would this be an effective or an inneffective ensemble method? (Briefly explain.)
This would not be an effective ensemble method, because since we have random initialization, it is possible that the same clusters are given different labels in the different bootstraps. Hence, we cannot simply choose the y_i that is most common.

11. What does minimizing $f(w) = ||Xw - y||_1 + \lambda||w||^2$ correspond to in the MLE/MAP view?
Finding the MAP estimate with a Laplacian likelihood, $y_i \sim \mathcal{L}(w^\top x_i, 1)$, and a Gaussian prior, $w \sim \mathcal{N}(0, \frac{1}{2\lambda})$.

12. If you perform MLE with a Gaussian likelihood, would the coefficient $\hat{w}$ change if you standardized the features? Would the predictions $\hat{y}^i$ change? Which of these would change if you perform MAP estimation with a Gaussian likelihood and Gaussian prior.
For MLE, coefficients would change but the predictions will not change. For MAP estimation, both the coefficient and the predictions will change.

13. Do you expect the resulting loss from running NMF to be lower, higher, or the same as the loss from running PCA on the same data? Briefly justify your answer.
The resulting loss from PCA is likely lower than the loss from NMF because PCA results in negative values.

14. Suppose we need to multiply a huge number of matrices to compute a product like $A_1 A_2 A_3 \cdots A_k$. The matrices have wildly-different sizes so the order of multiplication will affect the runtime. For

example, $A_1(A_2A_3)$ may be faster to compute than $(A_1A_2)A_3$. Describe (at a high level) an $O(k^3)$-time algorithm that finds the lowest-cost order to multiply the matrices.
We can use dynamic programming with memoization to find the minimum number of multiplications needed in $O(k^3)$ time, and then backtrack the solution to find the optimal order to perform the multiplications.

15. You have a supervised learning dataset $\{X, y\}$. You fit a 1-hidden-layer neural network using stochastic gradient descent to minimize the squared error, that makes predictions of the form $\hat{y}^i = v^\top W x^i$ where W and v are the parameters. Explain why or why not this neural network can achieve a better training accuracy than the basic linear regression model $\hat{y}^i = w^\top x^i$.
A neural network can model nonlinear functions, meaning it can form a very tight fit to the training data. Assuming that the data does not lie on a line/plane/hyperplane, it is likely to perform much better than the basic linear regression model w/ respect to the training accuracy.

2 Coding Questions

If you have not previously used Julia, there is a list of useful Julia commands (and syntax) among the list of notes on the course webpage.

2.1 Regularization and Hyper-Parameter Tuning

Download *a1.zip* from the course webpage, and start Julia (latest version) in a directory containing the extracted files. If you run the script *example_nonLinear* (from Julia's REPL), it will:

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

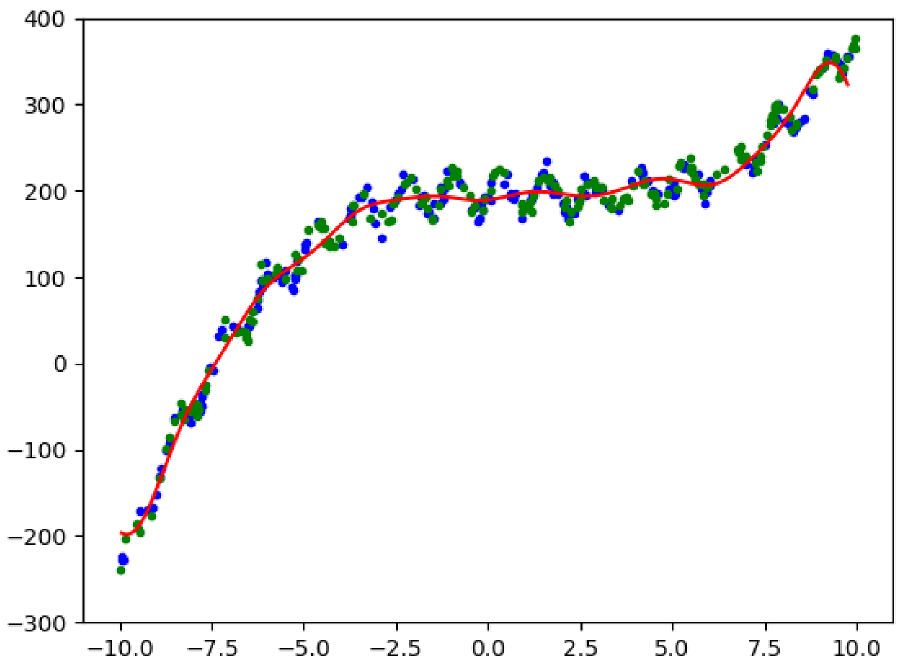
This script uses the *JLD* package to load the data and the *PyPlot* package to make the plot. If you have not previously used these packages, they can be installed using:

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

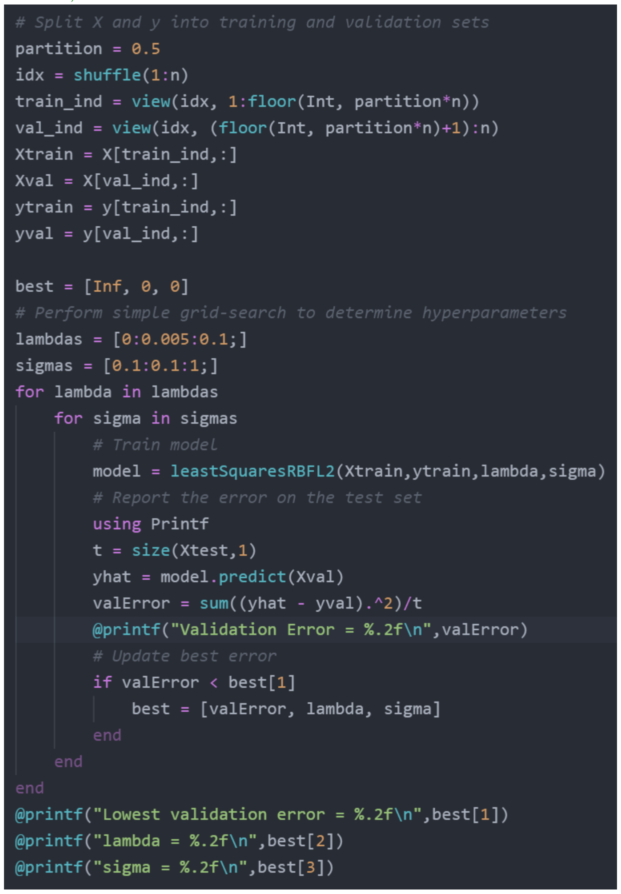
Unfortunately, this is not a great model of the data, and the figure shows that a linear model is probably not suitable.

1. Write a function called *leastSquaresRBFL2* that implements *least squares using Gaussian radial basis functions (RBFs) and L2-regularization.*
You should start from the *leastSquares* function and use the same conventions: n refers to the number of training examples, d refers to the number of features, X refers to the data matrix, y refers to the targets, Z refers to the data matrix after the change of basis, and so on. Note that you'll have to add two additional input arguments (λ for the regularization parameter and σ for the Gaussian RBF variance) compared to the *leastSquares* function. To make your code easier to understand/debug, you may want to define a new function *rbfBasis* which computes the Gaussian RBFs for a given training set, testing set, and σ value. Hand in your function and the plot generated with $\lambda = 1$ and $\sigma = 1$.
Note: the *distancesSquared* function in *misc.jl* is a vectorized way to quickly compute the squared Euclidean distance between all pairs of rows in two matrices.

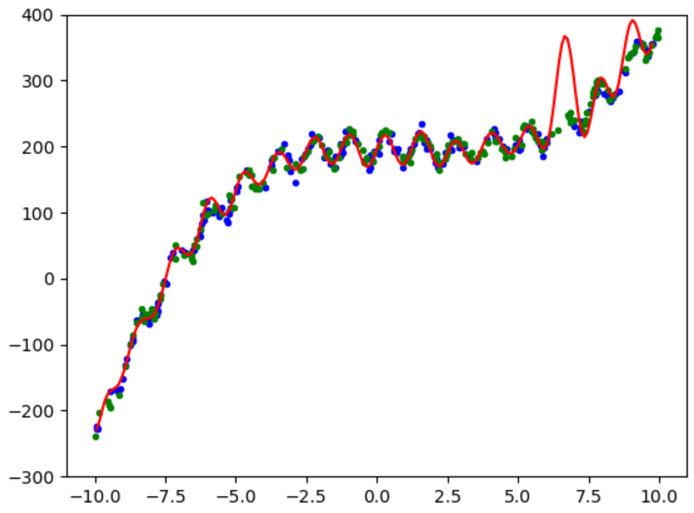
```
function rbfBasis(Xtilde,X,sigma)
    # Add bias column
    ntilde = size(Xtilde,1)
    Ztilde = [ones(ntilde,1) Xtilde]
    n = size(X,1)
    Z = [ones(n,1) X]
    # Calculate kernal using Gaussian radial basis function
    K = exp.(-distancesSquared(Ztilde, Z)./(2*sigma*sigma))
    return K
end

function leastSquaresRBFL2(X,y,lambda,sigma)
    # Find regression weights w/ L2 regularization
    K = rbfBasis(X,sigma)
    w = (K'*K + lambda*I)\(K'*y)
    # Make linear prediction function
    predict(Xtilde) = rbfBasis(Xtilde,X,sigma)*w
    # Return model
    return LinearModel(predict,w)
end
```



2. Modify the script to split the training data into a "train" and "validation" set (you can use half the examples for training and half for validation), and use these to select λ and σ. Hand in your modified script and the plot you obtain with the best values of λ and σ.
   $\lambda = 0$, $\sigma = 0.9$



```
# Split X and y into training and validation sets
partition = 0.5
idx = shuffle(1:n)
train_ind = view(idx, 1:floor(Int, partition*n))
val_ind = view(idx, (floor(Int, partition*n)+1):n)
Xtrain = X[train_ind,:]
Xval = X[val_ind,:]
ytrain = y[train_ind,:]
yval = y[val_ind,:]

best = [Inf, 0, 0]
# Perform simple grid-search to determine hyperparameters
lambdas = [0:0.005:0.1;]
sigmas = [0.1:0.1:1;]
for lambda in lambdas
    for sigma in sigmas
        # Train model
        model = leastSquaresRBFL2(Xtrain,ytrain,lambda,sigma)
        # Report the error on the test set
        using Printf
        t = size(Xtest,1)
        yhat = model.predict(Xval)
        valError = sum((yhat - yval).^2)/t
        @printf("Validation Error = %.2f\n",valError)
        # Update best error
        if valError < best[1]
            best = [valError, lambda, sigma]
        end
    end
end
@printf("Lowest validation error = %.2f\n",best[1])
@printf("lambda = %.2f\n",best[2])
@printf("sigma = %.2f\n",best[3])
```



3. There are reasons why this dataset is particularly well-suited to Gaussian RBFs are that (i) the period of the oscillations stays constant and (ii) we have evenly sampled the training data across its domain. If either of these assumptions are violated, the performance with our Gaussian RBFs might be much worse. Consider a scenario where either (i) or (ii) is violated, and describe a way that you could address this problem.
Let us imagine that (i) is violated, such that the period of the oscillations varies. We can use piecewise regression, combined with the Gaussian RBF, to model different periods of oscillations. In order for the piecewise regression to perform well, we must perform a hyperparameter search on where splines should be located (i.e. where in the data does the period of oscillation change).

2.2 Multi-Class Logistic Regression

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

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

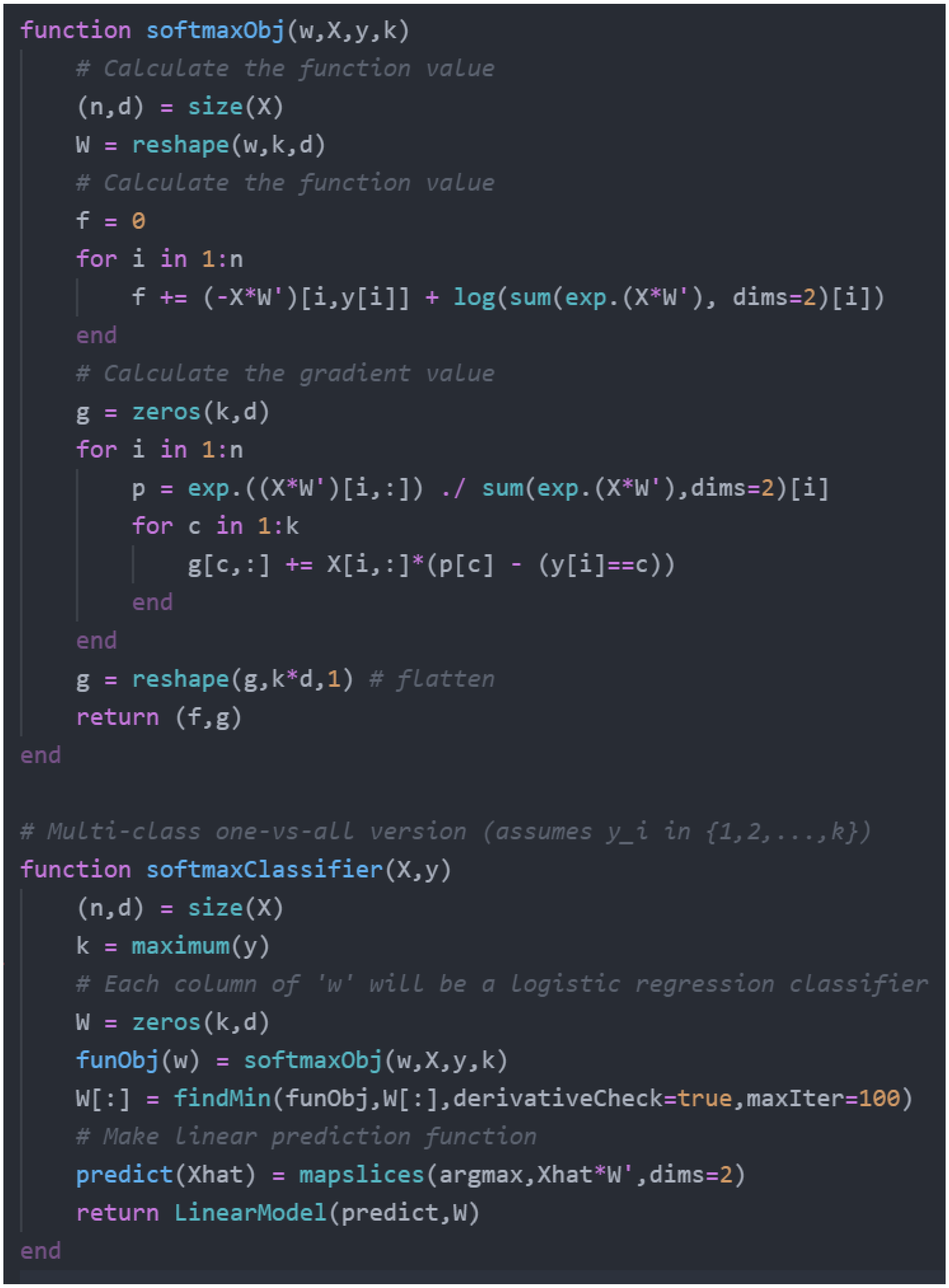
$$p(y^i|W, x^i) = \frac{\exp(w_{y^i}^\top x^i)}{\sum_{c=1}^k \exp(w_c^\top x^i)}.$$

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

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

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

Hint: you can use the *derivativeCheck* option when calling *findMin* to help you debug the gradient of the softmax loss. Also, note that the *findMin* function treats the parameters as a vector (you may want to use *reshape* when writing the softmax objective).
Validation error: 0.026



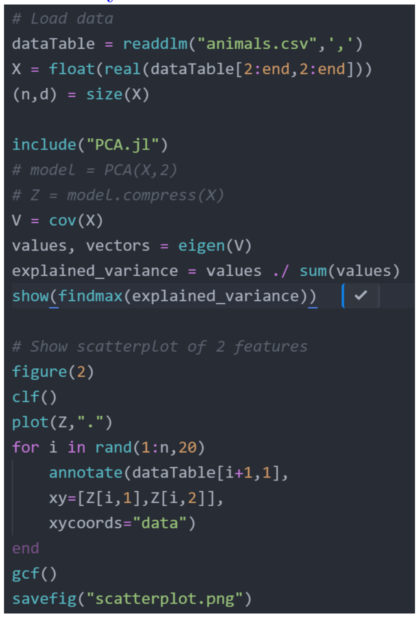
```
function softmaxObj(w,X,y,k)
    # Calculate the function value
    (n,d) = size(X)
    W = reshape(w,k,d)
    # Calculate the function value
    f = 0
    for i in 1:n
        f += (-X*W')[i,y[i]] + log(sum(exp.(X*W'), dims=2)[i])
    end
    # Calculate the gradient value
    g = zeros(k,d)
    for i in 1:n
        p = exp.((X*W')[i,:]) ./ sum(exp.(X*W'),dims=2)[i]
        for c in 1:k
            g[c,:] += X[i,:]*(p[c] - (y[i]==c))
        end
    end
    g = reshape(g,k*d,1) # flatten
    return (f,g)
end

# Multi-class one-vs-all version (assumes y_i in {1,2,...,k})
function softmaxClassifier(X,y)
    (n,d) = size(X)
    k = maximum(y)
    # Each column of 'w' will be a logistic regression classifier
    W = zeros(k,d)
    funObj(w) = softmaxObj(w,X,y,k)
    W[:] = findMin(funObj,W[:],derivativeCheck=true,maxIter=100)
    # Make linear prediction function
    predict(Xhat) = mapslices(argmax,Xhat*W',dims=2)
    return LinearModel(predict,W)
end
```

2.3 Principal Component Analysis

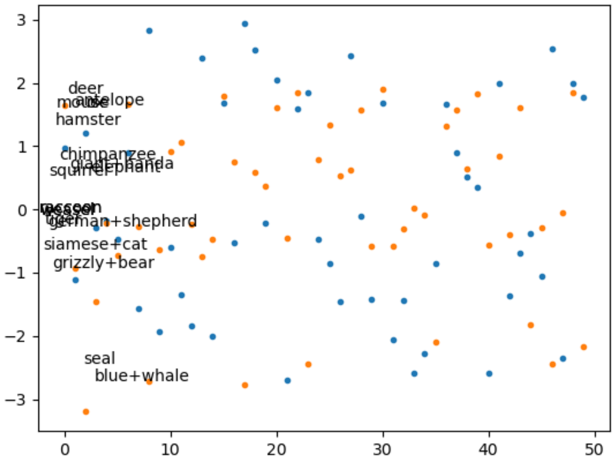
The script *example_PCA* will load a dataset containing 50 examples, each representing an animal. The 85 features are traits of these animals. The script gives two unsatisfying visualizations of it. First it shows a plot of the matrix entries, which has too much information and thus gives little insight into the relationships between the animals. Next it shows a scatterplot based on two random features and displays the name of 10 randomly-chosen animals. Because of the binary features even a scatterplot matrix shows us almost nothing about the data.

The function *PCA* applies the classic PCA method (orthogonal bases via SVD) for a given k. Using this function, modify the demo so that the scatterplot uses the latent features z_i from the PCA model with $k = 2$. Make a scatterplot of the two columns in Z, and use the *annotate* function to label a bunch of the points in the scatterplot.

1. Hand in your modified demo and the scatterplot.



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

include("PCA.jl")
# model = PCA(X,2)
# Z = model.compress(X)
V = cov(X)
values, vectors = eigen(V)
explained_variance = values ./ sum(values)
show(findmax(explained_variance))  ✓

# Show scatterplot of 2 features
figure(2)
clf()
plot(Z,".")
for i in rand(1:n,20)
    annotate(dataTable[i+1,1],
    xy=[Z[i,1],Z[i,2]],
    xycoords="data")
end
gcf()
savefig("scatterplot.png")
```



2. 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 trait in the *dataTable*).
domestic

3. Which trait of the animals has the largest influence (absolute value) on the second principal component?
nestspot

4. How much of the variance in X (after centering) is explained by our two-dimensional representation from the previous question?
Note: you can compute the Frobenius norm of a matrix using the function *norm*. Also, note that the "variance explained" formula from CPSC 340 assumes that X is already centered.
32.3%

5. How many PCs are required to explain 50% of the variance in the data?
5 principle components

3 Calculation Questions

3.1 Minimizing Strictly-Convex Quadratic Functions

Solve for the minimizer w of the below strictly-convex quadratic functions:

1. $f(w) = \frac{1}{2}w^\top \Lambda w + u^\top w + \lambda$ (general quadratic).

$$f(w) = \frac{1}{2}\sum_{j=1}^{d} \Lambda_{jj} w_j^2 + \sum_{j=1}^{d} u_j w_j + \lambda$$

$$\nabla f(w) = 2\Lambda w + u = 0 \longrightarrow \Lambda w = -\frac{1}{2}u \longrightarrow w = -\frac{1}{2}\Lambda^{-1}u$$

2. $f(w) = \frac{1}{2}(Xw - y)^\top \Sigma^{-1}(Xw - y) + \frac{\lambda}{2}\|w\|^2$ (L2-regularized least squares with weight matrix Σ).

$$\begin{aligned} f(w) &= \frac{1}{2}\sum_{i=1}^{n} \frac{(w^\top x_i - y_i)^2}{\Sigma_{ii}} + \frac{\lambda}{2}\sum_{j=1}^{d} w_j^2 \\ &= \frac{1}{2}(\frac{(w^T x_1 - y_1)^2}{\Sigma_{1,1}} + \frac{(w^T x_2 - y_2)^2}{\Sigma_{2,2}} + ... + \frac{(w^T x_n - y_n)^2}{\Sigma_{n,n}} + \lambda(w_1^2 + w_2^2 + ... + w_d^2)) \\ &= \frac{1}{2}(\frac{(w_1 x_{1,1} + ... + w_d x_{1,d} - y_1)^2}{\Sigma_{1,1}} + \frac{(w_2 x_{2,2} + ... + w_d x_{2,d} - y_2)^2}{\Sigma_{2,2}} + ... + \frac{(w_n x_{n,n} + ... + w_d x_{n,d} - y_n)^2}{\Sigma_{n,n}} + \lambda(w_1^2 + \end{aligned}$$

$$\frac{\delta f}{\delta w_j} = \sum_{i=1}^{n} \frac{w^T(x_i - y_i)x_{ij}}{\Sigma_{ij}} + w_j$$

$$\nabla f = \begin{bmatrix} \sum_{i=1}^{n} \frac{(w^T x_i - y_i)x_{i1}}{\Sigma_{i1}} + w_1 \\ \vdots \\ \sum_{i=1}^{n} \frac{(w^T x_i - y_i)x_{id}}{\Sigma_{id}} + w_d \end{bmatrix} = X^T\Sigma^{-1}(Xw - y) + w = 0$$

$$w = (X^T\Sigma^{-1}y)(X^T\Sigma^{-1}X + 1)^{-1}$$

3. $f(w) = \frac{1}{2}\sum_{i=1}^{n} v_i(w^\top x^i - y^i)^2 + \frac{\lambda}{2}\|w - u\|^2$ (weighted least squares shrunk towards u).

$$\begin{aligned} f(w) &= \frac{1}{2}([v_1(\sum_{j=1}^{d} w_j x_{1j} - y_1)^2 + ... + v_n(\sum_{j=1}^{d} w_j x_{nj} - y_n)^2] + \lambda\sum_{j=1}^{d}(w_j - u_j)^2) \\ &= \frac{1}{2}([v_1(\sum_{j=1}^{d} w_j x_{1j} - y_1)^2 + ... + v_n(\sum_{j=1}^{d} w_j x_{nj} - y_n)^2] + \lambda\sum_{j=1}^{d}(w_j^2 - 2w_j u_j + u_j^2)) \end{aligned}$$

$$\frac{\delta f}{\delta w_1} = \sum_{i=1}^{n} v_i(w^T x_i - y_i)x_{i1} + \lambda(w_1 - u_1) \longrightarrow \frac{\delta f}{\delta w_j} = \sum_{i=1}^{n} v_i(w^T x_i - y_i)x_{ij} + \lambda(w_j - u_j)$$

$$\nabla f = \begin{bmatrix} \sum_{i=1}^{n} v_i(w^T x_i - y_i)x_{i1} + \lambda(w_1 - u_1) \\ \vdots \\ \sum_{i=1}^{n} v_i(w^T x_i - y_i)x_{id} + \lambda(w_d - u_d) \end{bmatrix} = X^T V(Xw - y) + \lambda(w - u) = 0$$

$$w = \frac{X^T V y + \lambda u}{X^T V X + \lambda}$$

Above we use our usual supervised learning notation. In addition, we assume that u is $d \times 1$ and v is $n \times 1$, λ is a positive scalar, and Σ and Λ are symmetric positive-definite matrices. You can use V as a diagonal matrix with v along the diagonal (with the v_i non-negative). You can use I as an identity matrix. Hint: positive-definite matrices are invertible.

3.2 MAP Estimation

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

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

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

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

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

1. Laplace likelihood (with a scale of 1) for each training example and Laplace prior centered at u with scale $1/\lambda$.

$$y^i \sim \mathcal{L}(w^\top x^i, 1), \quad w_j \sim \mathcal{L}\left(u, 1/\lambda\right),$$

where u is $d \times 1$.

$$\pi(w_j) = f(w_j) = \frac{\lambda}{2}exp(-\lambda|w_j - u|) \longrightarrow ln(\pi(w_j)) = ln(\frac{\lambda}{2}) - \lambda|w_j - u|$$

$$L = f(w^T x_i|1) = \prod_{i=1}^{n}[\frac{1}{2}exp(-|w^T x_i - y_i|)] \longrightarrow ln(L) = -nln(2) - \sum_{i=1}^{n}|w^T x_i - y_i|$$

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

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

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

$$\pi(w_j) = \sqrt{\frac{\lambda_j}{2\pi}}exp(-\frac{1}{2}(\lambda_j w_j)^2) \longrightarrow ln(\pi(w_j)) = ln\sqrt{\frac{\lambda_j}{2\pi}} - \frac{1}{2}(\lambda_j w_j)^2$$

$$L = \prod_{i=1}^{n}\frac{1}{\sqrt{2\pi}\sigma_i}exp(-\frac{1}{2}(\frac{w^T x_i - y_i}{\sigma_i})^2) \longrightarrow ln(L) = nln(\frac{1}{\sqrt{2\pi}\sigma_i}) - \frac{1}{2}\sum_{i=1}^{n}(\frac{w^T x_i - y_i}{\sigma_i})^2$$

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

3. Time-independent censored survival analysis likelihood with a uniform prior,

$$p(y^i, v^i | x^i, w) = \exp(v^i w^T x^i) \exp(-y^i \exp(w^T x^i)), \quad p(w_j) \propto \kappa$$

Here, y^i is a positive number giving the latest time that we observed patient i, and $v^i = 1$ if patient i has quit the study while $v^i = 0$ if they are still in it. For this question, you do not need to convert to matrix notation.

$$L(y_i, v_i) = \prod_{i=1}^{n} \exp(v_i w^T x_i) \exp(-y_i \exp(w^T x_i)) \longrightarrow ln(L) = \sum_{i=1}^{n} [v_i w^T x_i - y_i \exp(w^T x_i)]$$

$$f(w) = ln(L) + \sum_{j=1}^{d} lnp(w_j) = \sum_{i=1}^{n} [v_i w^T x_i - y_i \exp(w^T x_i)] + dln(\kappa)$$

[1]

3.3 Machine Learning Model Memory and Time Complexities

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

1. What is the storage space required for a naive Bayes classifier with binary features and k class labels? $O(2kn) \approx O(kn)$, since for each class k and each example n, there are 2 possible values.
2. What is the training time required for k-means clustering? You can use t as the number of iterations it takes to converge. $O(tknd)$
3. What is the training time for linear regression with Gaussian RBF features? You can use σ^2 as the variance of the Gaussian RBFs. $O(n^2 d + n^3)$
4. What is the storage space required for an L2-regularized linear regression model with a polynomial basis, where we have used the kernel trick? You can use λ as the regularization parameter and p as the degree of the polynomial. $O(n(k+p))$
5. What is the cost of trying to minimize the logistic regression loss by running t iterations of stochastic gradient descent? $O(ndt)$
6. What is the cost of forward propagation (computing one value $\hat{y}_i$) in a neural network (for regression) with 3 fully-connected hidden layers? Use k_1 as the number of hidden units in layer 1, k_2 as the number of hidden units in layer 2, and k_3 as the number of hidden units in layer 3. $O(dk_1 + k_1 k_2 + k_2 k_3 + k_3)$

3.4 Gradients and Hessians in Matrix Notation

Express the gradient $\nabla f(w)$ and Hessian $\nabla^2 f(w)$ of the following functions in matrix notation, simplifying as much as possible:

1. The quadratic function

$$f(w) = w^T u + u^T A w + \frac{\lambda}{2} w^T w + w^T A w,$$

[1]This likelihood can be used in regression settings to estimate survival times when some patients are still alive.

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

wher u is $d \times 1$ and A is $d \times d$ (not necessarily symmetric).

$$\nabla f(w) = u + u^\top A + \frac{\lambda}{2} w + (A + A^\top) w$$

$$\nabla^2 f(w) = 0 + 0 + \frac{\lambda}{2} + A + A^\top = \frac{\lambda}{2} + A + A^\top$$

2. L2-regularized weighted least squares with non-Euclidean quadratic regularizaiton,

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

where you can use V as a matrix with the v_i along the diagonal and Λ as a positive-definite $d \times d$ (symmetric) matrix with λ_{ij} in position (i, j).

$$f(w) = (Xw - y)^\top V (Xw - y) + \frac{1}{2} w^\top \Lambda w = (Xw - y)^\top (XVw - Vy) + \frac{1}{2} w^\top \Lambda w$$

$$\begin{aligned} \nabla f(w) &= X^\top V X w - X^\top V y + \frac{1}{2}(\Lambda + \Lambda^\top) w \\ &= X^\top V (Xw - y) + \frac{1}{2}(\Lambda + \Lambda^\top) w \\ \nabla^2 f(w) &= X^\top V X + \frac{1}{2}(\Lambda + \Lambda^\top) \end{aligned}$$

3. Weighted L2-regularized probit regression,

$$f(w) = -\sum_{i=1}^{n} \log p(y^i | x^i w) + \frac{1}{2} \sum_{j=1}^{d} u_j w_j^2.$$

where u is $d \times 1$, $y^i \in \{-1, +1\}$, and the likelihood of a single example i is given by

$$p(y^i | x^i, w) = \Phi(y^i w^T x^i).$$

where Φ is the cumulative distribution function (CDF) of the standard normal distribution.

$$f(w) = -\sum_{i=1}^{n} log(c_i) + \frac{1}{2} w^\top u w, \text{ and } M = \frac{p_i y^i}{c_i}$$

$$\nabla f(w) = -\sum_{i=1}^{n} \frac{p_i x^i y^i}{c_i} + \frac{1}{2} w^\top u = -\sum_{i=1}^{n} M_i x^i + \frac{1}{2} w^\top u = X^\top M + \frac{1}{2} w^\top u$$

$$\nabla^2 f(w) = -\sum_{i=1}^{n} \frac{p_i y^i}{c_i} (x^i w + \frac{p_i y^i}{c_i}) + \frac{1}{2} u$$

Hint: You can use the results from the linear and quadratic gradients and Hessians notes to simplify the derivations. You can use 0 to represent the zero vector or a matrix of zeroes and I to denote the identity matrix. It will help to convert the second question to matrix notation first. For the last question, it is useful to define a vector c containing the CDF $\Phi(y^i w^T x^i)$ as element c_i and a vector p containing the corresponding PDF as element p_i. For the probit question you'll need to define new vectors to express the gradient and Hessian in matrix notation (and remember the relationship between the PDF and CDF). As a sanity check, make sure that your results have the right dimension.

3.5 Norm Inequalities

Show that the following inequalities hold for vectors $w \in \mathbb{R}^d$, $u \in \mathbb{R}^d$, and $X \in \mathbb{R}^{n \times d}$:

1. $\|w\|_\infty \leq \|w\|_2 \leq \|w\|_1$ (relationship between decreasing p-norms)

$$\|w\|_\infty = max_j^d(w_j), \|w\|_2 = \sqrt{\sum_{j=1}^{d}(w_j)^2}, \|w\|_1 = \sum_{j=1}^{d}|w_j|$$

$$\|w\|_\infty \leq \|w\|_2 \equiv (max_j^d(w_j))^2 \leq w_1^2 + w_2^2 + ... + w_d^2$$

Evidently, the maximum of w must be included in w, which means $(max_j^d(w_j))^2 < \sum_{j=1}^{d}(w_j)^2$, and $\|w\|_\infty \leq \|w\|_2$.

$$\begin{aligned}\|w\|_2 \leq \|w\|_1 &\equiv \sum_{j=1}^{d}(w_j)^2 \leq (\sum_{j=1}^{d}|w_j|)^2 \\ &\equiv w_1^2 + w_2^2 + ... + w_d^2 \leq (|w_1| + |w_2| + ... + |w_d|)^2 \\ &\equiv w_1^2 + w_2^2 + ... + w_d^2 \leq (w_1^2 + w_2^2 + ... + w_d^2) + 2\sum_{i \neq j}|w_i||w_j| \\ &\equiv 0 \leq 2\sum_{i \neq j}|w_i||w_j|\end{aligned}$$

$2\sum_{i \neq j}|w_i||w_j| \geq 0$, so $\|w\|_2 \leq \|w\|_1$.

2. $\|w\|_1 \leq \sqrt{d}\|w\|_2 \leq d\|w\|_\infty$ (relationship between increasing p-norms)

$$\sqrt{d}\|w\|_2 = \sqrt{d\sum_{j=1}^{d}(w_j)^2}, d\|w\|_\infty = d * max_j^d(w_j)$$

$$\|w\|_1 \leq \sqrt{d}\|w\|_2 \equiv \|w_2\|_2 * \|sign(w)\|_2 * cos(\theta) \leq \sqrt{d}\|w\|_2$$

$$\begin{aligned}\|w_2\|_2 * \|sign(w)\|_2 * cos(\theta) &\leq \|w\|_2 * \|sign(w)\|_2 \\ &\leq \|sign(w)\|_2 = \sqrt{\sum_{i=1}^{d} \pm 1^2} = \sqrt{d}\end{aligned}$$

Since sign(w) is a vector that contains the signs of w's values (either 1 or -1), $\|w\|_2$ must equal $\sqrt{d}$, which is less than or equal to $\sqrt{d}\|w\|_2$. Therefore, $\|w\|_1 \leq \sqrt{d}\|w\|_2$.

$$\begin{aligned}\sqrt{d}\|w\|_2 \leq d\|w\|_\infty &\equiv d\sum_{j=1}^{d}(w_j)^2 \leq d^2 * max_j^d(w_j)^2 \\ &\equiv \sum_{j=1}^{d}(w_j)^2 \leq d * max_j^d(w_j)^2 \\ &\equiv w_1^2 + w_2^2 + ... + w_d^2 \leq d * w_{max}^2\end{aligned}$$

Given that all $w_j \neq w_{max}$ are strictly less than w_{max}, we know that $w_1^2 + w_2^2 + ... + w_d^2 \leq d * w_{max}^2$, and therefore $\sqrt{d}\|w\|_2 \leq d\|w\|_\infty$.

3. $\sqrt{m}\|w\|_2 \leq \|w\|_H \leq \sqrt{M}\|w\|_2$ (relationship between quadratic norm and Euclidean norm).

$$\text{Let } u = Q^\top w,\ H = Q\Lambda Q^\top, \text{ and } u^\top \Lambda u = \sum_{j=1}^{d} \lambda_{jj} u_j^2$$

$$\|w\|_H = \sqrt{w^\top H w} = \sqrt{w^\top Q\Lambda Q^\top w} = \sqrt{u^\top \Lambda u} = \sqrt{\sum_{j=1}^{d} \lambda_{jj} u_j^2}$$

$$\begin{aligned}\sqrt{m}\|w\|_2 \leq \|w\|_H &\equiv \sqrt{m\sum_{j=1}^{d} w_j^2} \leq \sqrt{\sum_{j=1}^{d} \lambda_{jj} u_j^2}\\ &\equiv m\sum_{j=1}^{d} w_j^2 \leq \sum_{j=1}^{d} \lambda_{jj} u_j^2\end{aligned}$$

Since m is the smallest eigenvalue of all λ_{jj}, $m\sum_{j=1}^{d} w_j^2$ must be smaller than $\sum_{j=1}^{d} \lambda_{jj} u_j^2$, and only equal to $\sum_{j=1}^{d} \lambda_{jj} u_j^2$ if all $\lambda_{jj} = m$.

$$\begin{aligned}\|w\|_H \leq \sqrt{M}\|w\|_2 &\equiv \sqrt{\sum_{j=1}^{d} \lambda_{jj} u_j^2} \leq \sqrt{M\sum_{j=1}^{d} w_j^2}\\ &\equiv \sum_{j=1}^{d} \lambda_{jj} u_j^2 \leq M\sum_{j=1}^{d} w_j^2\end{aligned}$$

Since M is the largest eigenvalue of all λ_{jj}, $M\sum_{j=1}^{d} w_j^2$ must be larger than $\sum_{j=1}^{d} \lambda_{jj} u_j^2$, and only equal to $\sum_{j=1}^{d} \lambda_{jj} u_j^2$ if all $\lambda_{jj} = M$.

You should use the definitions of the norms, but should not use the known equivalences between these norms (since these are the things you are trying to prove). Hint: for many of these it's easier if you work with squared values (and you may need to "complete the square"). Beyond non-negativity of norms, it may also help to use the Cauchy-Schwartz inequality, and/or to use that $\|x\|_1 = x^\top \text{sign}(x)$. We've used M as the largest eigenvalue of the (positive-definite) matrix H and m as the smallest eigenvalue, so $mI \preceq H \preceq MI$.