Homework 7

1. *Fitting a multi-class classifier.* In `multi_class.json`, you will find a $300 \times 30$ matrix $U_{\text{train}}$ and a 300-vector $v_{\text{train}}$ consisting of raw training input and output data, and a $300 \times 30$ matrix $U_{\text{test}}$ and a 300-vector $v_{\text{test}}$ consisting of raw test input and output data, respectively. We will work with $x = \phi(u) = (1, u)$.

In `multi_logistic.jl` we have also provided you with a function

\[
\text{multi_logistic}(X, Y, \text{reps})
\]

This function takes in input/output data $X$ and $Y$, and an array of representations of our embeddings $\text{reps}$. That is, the $k$th element of $\text{reps}$ is $\psi(v_k)$, for $k = 1, \ldots, K$, and $K$ is the number of labels.

The function outputs the *parameterized predictor* $g$ for the multi-class classifier. (Since $g$ is a predictor, its argument is simply $x$; to get a prediction $\hat{y}$ from a data input $x$, use $\hat{y} = g(x)$.) You must include the Flux and LinearAlgebra Julia packages in your code in order to utilize this function.

(a) Inspect $v_{\text{train}}$ and $v_{\text{test}}$. What is $\mathcal{V}$, the label set? What is $K$, the number of labels? Report the number of instances of each label in the training set. Do the same for the test set.

(b) Propose a simple embedding $\psi: \mathbb{R} \rightarrow \mathbb{R}^K$ for your raw output data $v$, so that $y = \psi(v)$.

(c) Apply your embedding from part (b) to both the training and test raw output data. Using `multi_logistic.jl`, fit a multi-class classifier to the training data. Use a nearest neighbor un-embedding to un-embed your data. Report the confusion matrices of your classifier on the training and test set. Also report the overall training error rate and the overall test error rate.

*Julia hints.* `unique(z)` returns the number of unique elements in an array $z$. For the nearest neighbor unembedding $\psi^\dagger(\hat{y})$, you may use

\[
\text{psiinv}(\hat{y}) = \text{reps}[\text{argmin}([\text{norm}(\hat{y} - \text{reps}[k]) \text{ for } k=1:\text{length}(\text{reps})])],
\]

where $\text{reps}$ is an array whose $k$th element is $\psi(v_k)$, for $k = 1, \ldots, K$.

Solution.

(a) $\mathcal{V} = \{-1, 0, 1\}$, so $K = 3$. In the training set, $v = -1$ occurs 118 times, $v = 0$ occurs 68 times, and $v = 1$ occurs 114 times. In the test set, $v = -1$ occurs 121 times, $v = 0$ occurs 65 times, and $v = 1$ occurs 114 times.
(b) We propose the embedding
\[ \psi(v) = \begin{cases} 
(1, 0, 0) & v = -1 \\
(0, 1, 0) & v = 0 \\
(0, 0, 1) & v = 1.
\end{cases} \]

(c) The training confusion matrix is
\[ C_{\text{train}} = \begin{bmatrix} 111 & 5 & 0 \\
3 & 58 & 6 \\
0 & 5 & 112 \end{bmatrix}. \]

The overall training error rate is approximately 0.063. Depending on random train/test split the error rate may be higher. The test confusion matrix is
\[ C_{\text{test}} = \begin{bmatrix} 101 & 13 & 1 \\
13 & 37 & 10 \\
0 & 15 & 110 \end{bmatrix}. \]

The overall test error rate is approximately 0.173. Depending on random train/test split the error rate may be higher.
In [3]:

```julia
using LinearAlgebra
using Random
using Flux

include("readclassjson.jl")
include("writeclassjson.jl")
include("multi_logistic.jl")

import PyCall
import PyPlot; const plt = PyPlot; plt.plt.style.use("seaborn")

Random.seed!(0)
```

Out[3]: MersenneTwister(0)

In [ ]:

```julia
n = 600

# X = randn(n,d)
# y = []
# for i=1:n
#   offset = 1.2
#   linsum = offset + sum(X[i,1:7]) + 0.2*sum(X[i,10:17]) - 0.7*sum(X[i,20:27]) + 0.6*randn()
#   if linsum - offset < -1
#     push!(y, -1)
#   elseif -1 <= linsum - offset <= 1
#     push!(y, 0)
#   elseif 1 < linsum - offset
#     push!(y, 1)
#   end
# end
# y = convert(Array{Int64,1}, y)

# global X_train = X[1:n÷2, :]
# X_test = X[n÷2+1:n, :]

# global y_train = y[1:n÷2]
# y_test = y[n÷2+1:n]

# Data = Dict()
# Data["U_train"] = X_train
# Data["v_train"] = y_train
# Data["U_test"] = X_test
# Data["v_test"] = y_test
# writeclassjson(Data, "multi_class.json")

Data = readclassjson("multi_class.json")

X_train = Data["U_train"]
X_test = Data["U_test"]
y_train = Data["v_train"]
y_test = Data["v_test"]
```
In [5]:
# part a
V = unique(y_train)
K = length(V)
println(V, " ", K)
println("X_TRAIN")
println(sum(y_train .== -1), " ", sum(y_train .== 0), " ", sum(y_train .== 1))
println("X_TEST")
println(sum(y_test .== -1), " ", sum(y_test .== 0), " ", sum(y_test .== 1))

[1, 0, -1] 3
X_TRAIN
118 68 114
X_TEST
121 65 114

In [6]:
function embedy(V, classes, reps)
    classnum(v) = findfirst(x -> x == v[1], classes)
    Y = ([reps[classnum(v)] for v in eachrow(V)])
    return vcat(Y'...)
end

Out[6]: embedy (generic function with 1 method)

In [ ]:
classes = unique(y_train)
K = length(classes)
function canonical(k,K)
    x = zeros(K)
    x[k] = 1
    return x
end
reps = [canonical(k,K) for k=1:K]
predicty, theta = multi_logistic([ones(n÷2) X_train], embedy(y_train, classes, reps), reps)
In [8]:
psiinv(yhat) = classes[argmin([norm(yhat-reps[k]) for k=1:K])]
G(u) = psiinv(predicty(u))

Xtr = [ones(n÷2) X_train]
Xte = [ones(n÷2) X_test]

# Xtr = X_train
# Xte = X_test

preds = [G(Xtr[i,:]) for i=1:n÷2]
acc_train = sum(preds == y_train)

preds = [G((Xte[i,:])) for i=1:n÷2]
acc_test = sum(preds == y_test)

println(acc_train, " ", acc_test)

281 248

In [9]:
function confusion(G, U, V, classes)
    # confusion
    K = length(classes)
    C=zeros(K,K)
    for i=1:K
        for j=1:K
            C[i,j] = sum([ (G(u) == classes[i] && v == classes[j]) for (u,v) in zip(U,V)])
        end
    end
    println("Confusion matrix:")
    display(C)
    return C
end

Out[9]: confusion (generic function with 1 method)

In [10]:
C = confusion(G, [Xtr[i,:] for i in 1:n÷2], y_train, classes)
overall_train_error_rate = sum([C[i,j] for i=1:K for j=1:K if i != j])/(n÷2)

3x3 Matrix{Float64}:
  111.0  5.0  0.0
  3.0  58.0  6.0
  0.0  5.0  112.0

Confusion matrix:

Out[10]: 0.06333333333333334
In [11]:
C = confusion(G, [Xte[i, :] for i in 1:n÷2], y_test, classes)
overall_test_error_rate = sum([C[i, j] for i=1:K for j=1:K if i != j])/(n÷2)

3×3 Matrix{Float64}:
 101.0  13.0    1.0
 13.0  37.0   10.0
 0.0  15.0  110.0

Confusion matrix:

Out[11]:
0.17333333333333334
2. Predicting Covid-19 infection based on symptoms.

In this problem we would like to build a binary classifier for a highly relevant problem: given a list of symptoms, predict whether or not a patient has Covid-19.

In `covid_symptoms.json` you will find a matrix $U$ with 20 binary features (answers to yes/no questions) and a vector $v$ with the binary classification output: 1 if the patient was diagnosed with Covid-19 and -1 if they were not.

We have also provided a function in `classification_regression_fit.jl`.

```julia
classification_regression_fit(X, Y, l, r, lambda, kappa; numiters=40)
```

This function returns $\theta^*$, the parameter vector for a linear classifier:

$$
\theta^* \leftarrow \text{minimize } l(\hat{y}, y) + \lambda r(\theta)
$$

This is almost the same function you used a few weeks ago, with the addition of $\kappa$: the parameter for your binary classification loss. It takes in input/output data $X$ and $Y$, a loss function $l(\hat{y}, y)$, a regularizer $r(\theta)$, $\lambda$, $\kappa$, and optionally, the number of epochs Flux should run. You will have to import `LinearAlgebra` and `Flux` to run the function.

(a) In binary classification we can use the Neyman-Pearson metric with parameter $\kappa$:

$$
\kappa E_{fn} + E_{fp}
$$

$\kappa$ expresses our relative dislike for mistaking a positive example for a negative one. In this context (predicting Covid-19 infection), which is more dangerous: a false positive or a false negative? Should $\kappa$ be less than 1 or greater than 1? Provide a short justification explaining your answer.

(b) First, add a column of ones to $U$ (as in previous homeworks). We will use a random 90/10 train/test split. Partition your data into $U_{\text{train}}$, $v_{\text{train}}$, $U_{\text{test}}$ and $v_{\text{test}}$.

We will use SVM. Define the hinge loss:

$$
\ell_{\text{hinge}}(\hat{y}, y) = \begin{cases} 
(1 + \hat{y})_+ & \text{if } y = -1 \\
\kappa(1 - \hat{y})_+ & \text{if } y = 1 
\end{cases}
$$

and L2 regularizer (you can exclude the $\theta_i$ corresponding to your column of ones, although it’s not required).

$$
r(\theta) = \|\theta\|_2^2
$$

You can set $\lambda = 0.01$ or experiment with different values; the focus of this problem is understanding how $\kappa$ changes the false negative and false positive rate of our SVM classifier.
(b) Figure 1 False negatives and false positives. As expected, more false positives = fewer false negatives. Some students regularized the whole $\theta$ while others did $\theta_{2:K}$. If they regularized the whole $\theta$ we found the first 2 values may be very similar. This is OK.

(c) For $\kappa$ between 0.01 and 10 (see the hint), use `classification_regression_fit` to compute the predictor $\theta$. Find the accuracy and number of false negatives and positives for the test set. Plot the false negatives and false positives on one graph (with $\kappa$ on the x axis).
Plot the accuracy on a separate graph (with $\kappa$ on the x axis).

(d) Evaluate your results from part (c) and choose the “best” classifier (for example, the one with highest accuracy or fewest false negatives). Report the accuracy and confusion matrix of the “best” classifier you chose.

*Hint:* Use `kappas = 10 .^ range(-2,1,length=10)` to compute your vector of $\kappa$ values.

*Hint:* `sign.(y_hat)` will convert a vector $y\_hat$ of floating-point values to a prediction of -1 or +1.

*Hint:* Using a logarithmic x axis for these graphs will help you see the trend. To plot $x$ and $y$ with a log-space x axis, use `plot(x, y, xaxis=:log)`.

**Solution.**

(a) A reasonable answer is we want to penalize false negatives more than false positives, so $\kappa > 1$ is preferable.

(c) The best predictor we found (results may vary) is:

**Kappa:** 1.0
Figure 2 Accuracy. Some students regularized the whole $\theta$ while others did $\theta_{2,k}$. If they regularized the whole $\theta$ we found the first 2 accuracy values may be very similar. This is OK.

Accuracy: 0.9799270072992701
True neg: 95
True pos: 442
False neg: 0
False pos: 11

Some students also found $\kappa = 2.15$ to be the best value.
```julia
In [1]:
using LinearAlgebra
using Random
using Flux
using Flux: logitcrossentropy, onehot, onecold
using Plots
include("classification_regression_fit.jl")

Out[1]: classification_regression_fit (generic function with 1 method)

In [10]:
#using CSV, JSON
#include("writeclassjson.jl")
#file = CSV.File("Covid Dataset.csv")
# fun fact: this data was probably generated, not real. It came from Kaggle

# row = file[1]

#k = 20
#N = 5434
#U_data = zeros(N, k)
#v_data = zeros(N)
i=1
#for row in file
#    tmp = values(row)
#    U_data[i, :] = [answer == "No" ? 0.0 : 1.0 for answer in tmp[1:end-1]]
#    v_data[i] = tmp[end] == "No" ? -1.0 : 1.0
#    i += 1
#end

#Data = Dict()
#Data["U"] = U_data
#Data["v"] = v_data
#writeclassjson(Data, "covid_symptoms.json")

#include("readclassjson.jl")
data = readclassjson("covid_symptoms.json")
U_data = data["U"]
v_data = data["v"]
print("Loaded data")

Loaded data
```
In [11]:
N = size(U_data)[1]
indices = shuffle(1:N)
N_split = div(N,10)*9 # 90% split

U_train = [ones(N_split) U_data[indices[1:N_split]],:]
U_test = [ones(N+1-N_split) U_data[indices[N_split:end]],:]

v_train = v_data[indices[1:N_split]]
v_test = v_data[indices[N_split:end]]

N_test = size(v_test)[1]
println("Out of \$(N)\) samples, \$(N_split)\) used for training. \$(N-N_split)\) used for test."

Out of 5434 samples, 4887 used for training. 547 used for test.

In [23]:
#function logistic_loss(yhat, y, κ=2.0)
  #   if y == 1.0
  #     return κ*log.(1.0 .+ exp.(-yhat))
  #   elseif y == -1.0
  #     return log.(1.0 .+ exp.(yhat))
  #   end
#end

  function hinge_loss(yhat, y, κ=2.0)
    if y == 1.0
      return κ*max.(1.0 .- yhat, 0.0)
    elseif y == -1.0
      return max.(1.0 + yhat, 0.0)
    end
  end

  function l2_reg(theta)
    return norm(theta[2:end], 2)^2
  end

Out[23]: l2_reg (generic function with 1 method)
In [24]: # Alternative (brute force) way of generating the confusion matrix.
function confusion_mx(y_pred, y)
    tn, fn, tp, fp = 0, 0, 0
    for (yhat, y) in zip(y_pred, y)
        if yhat == y
            if y == -1
                tn += 1
            elseif y == 1
                tp += 1
        elseif y == -1 && yhat == 1
            fp += 1
        elseif y == 1 && yhat == -1
            fn += 1
        else
            print("Unhandled!")
        end
    end
    println("True neg: $(tn)")
    println("True pos: $(tp)")
    println("False neg: $(fn)")
    println("False pos: $(fp)")
    println("$(tp), $(fn)\n$(fp), $(tn)")
    return tn, tp, fn, fp
end

Out[24]: confusion_mx (generic function with 1 method)

In [ ]: fps = zeros(10) # false positives
fns = zeros(10) # false negatives
κs = 10 .^ range(-2,1,length=10)
accuracy = zeros(10)
thetas = []

i = 1
for κ in κs
    theta_svm = classification_regression_fit(U_train, v_train, hinge_loss, l2_reg, 0.01, κ, numiters=50)
    push!(thetas, theta_svm)
preds = sign.(U_test*theta_svm)
    tn, tp, fn, fp = confusion_mx(preds, v_test)
    fps[i] = fp
    fns[i] = fn
    accuracy[i] = sum(preds .== v_test)/N_test
    i += 1
end
In [26]:
scatter(ks, fps, label="False positives", color="blue", xaxis=:log, size=(400,300))
plot!(ks, fps, color="blue")
scatter!(ks, fns, label="False negatives", color="orange")
plot!(ks, fns, color="orange")
title!("False negatives and positives")
xlabel("\(\kappa\)"); ylabel("Number of FPs and FNs")
#savefig("fn_fp.png")

Out[26]:

Out[27]:

In [27]:
scatter(ks, accuracy*100.0, color="blue", xaxis=:log, size=(400,300))
plot!(ks, accuracy*100.0, color="blue")
title!("Accuracy")
xlabel("\(\kappa\)"); ylabel("% Correct")
#savefig("accuracy.png")

Out[27]:
In [29]: theta_best = thetas[argmax(accuracy)]
println("Kappa: ", κs[argmax(accuracy)])
preds = sign.(U_test*theta_best)
println("Accuracy: $(sum(preds .== v_test)/N_test)")
confusion_mx(preds, v_test)

Kappa: 1.0
Accuracy: 0.9799270072992701
True neg: 84
True pos: 453
False neg: 2
False pos: 9
453, 2
9, 84

Out[29]: (84, 453, 2, 9)


<table>
<thead>
<tr>
<th>Prediction</th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>TP</td>
<td>FN</td>
</tr>
<tr>
<td>Negative</td>
<td>FP</td>
<td>TN</td>
</tr>
</tbody>
</table>

This is telling us we have lots of false positives and no false negatives. We want to be cautious and avoid predicting someone doesn't have covid-19 if they actually do, so this is preferable to having many false negatives.