Final Homework

December 16, 2022

1 Question 4

1.1 This problem involves the OJ data set which is part of the ISLR2 package.

1.1.1 *(a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.*

Loading the libraries

[3]:
```
install.packages('ISLR')
install.packages('tidyverse')
install.packages('ggthemes')
install.packages('caret')
install.packages('e1071')

library(ISLR);
library(tidyverse);
library(ggthemes)
library(caret);
library(e1071)
```

```
The downloaded binary packages are in
/var/folders/ch/lqq67w6x5px6fcc9cnbp457m0000gn/T//RtmpgcVXgN/downloaded_packages

The downloaded binary packages are in
/var/folders/ch/lqq67w6x5px6fcc9cnbp457m0000gn/T//RtmpgcVXgN/downloaded_packages

The downloaded binary packages are in
/var/folders/ch/lqq67w6x5px6fcc9cnbp457m0000gn/T//RtmpgcVXgN/downloaded_packages

The downloaded binary packages are in
/var/folders/ch/lqq67w6x5px6fcc9cnbp457m0000gn/T//RtmpgcVXgN/downloaded_packages

The downloaded binary packages are in
/var/folders/ch/lqq67w6x5px6fcc9cnbp457m0000gn/T//RtmpgcVXgN/downloaded_packages
```

[6]:
```
set.seed(123)
```

```
data('OJ')

sample <- sample(nrow(OJ), 800, replace = FALSE)

train <- OJ[sample,]
test <- OJ[-sample,]
```

[7]:
```
head(train,5)
```

A data.frame: 5 × 18

	Purchase <fct>	WeekofPurchase <dbl>	StoreID <dbl>	PriceCH <dbl>	PriceMM <dbl>	DiscCH <dbl>	DiscMM <dbl>
415	MM	238	3	1.79	2.09	0.0	0.0
463	CH	228	7	1.69	1.69	0.0	0.0
179	CH	244	7	1.86	2.09	0.0	0.2
526	MM	263	1	1.76	1.99	0.0	0.4
195	CH	271	4	1.99	2.09	0.1	0.4

[8]:
```
head(test,5)
```

A data.frame: 5 × 18

	Purchase <fct>	WeekofPurchase <dbl>	StoreID <dbl>	PriceCH <dbl>	PriceMM <dbl>	DiscCH <dbl>	DiscMM <dbl>
1	CH	237	1	1.75	1.99	0.00	0.0
3	CH	245	1	1.86	2.09	0.17	0.0
7	CH	232	7	1.69	1.99	0.00	0.4
9	CH	235	7	1.75	1.99	0.00	0.4
12	CH	263	7	1.86	2.13	0.27	0.0

1.1.2 *(b) Fit a support vector classifier to the training data using cost = 0.01, with Purchase as the response and the other variables as predictors. Use the summary() funcction to produce summary statistics, and describe the results obtained.*

[36]:
```
set.seed(123)
svm_linear <- svm(Purchase ~ ., kernel='linear',  data=train, cost=0.01)
summary(svm_linear)
```

```
Call:
svm(formula = Purchase ~ ., data = train, kernel = "linear", cost = 0.01)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01

Number of Support Vectors:  442
```

```
 ( 220 222 )


Number of Classes:  2

Levels:
 CH MM
```

Out of 800 training points, the Support Vector classifier generates 442 support vectors. 220 of these are under level CH, and the remaining 222 fall under level MM.

1.1.3 *(c) What are the training and test error rates?*

[37]:
```
pred_train_data <- predict(svm_linear, train)
table(train$Purchase, pred_train_data)

cat("Train Error is:", 100 * mean(pred_train_data != train$Purchase), "%\n")
```

```
    pred_train_data
      CH  MM
  CH 426  61
  MM  71 242
```

```
Train Error is: 16.5 %
```

[88]:
```
pred_test_data <- predict(svm_linear, test)
table(test$Purchase, pred_test_data)

cat("Test Error is:", 100 * mean(pred_test_data != test$Purchase), "%\n")
```

```
    pred_test_data
      CH  MM
  CH 145  21
  MM  27  77
```

```
Test Error is: 17.77778 %
```

1.1.4 *(d) Use the tune() function to select an optimal cost. Consider values in the range 0.01 to 10.*

[40]:
```
set.seed(123)

tune_data <- tune(svm, Purchase ~ ., data = train, kernel='linear', ranges =␣
 ↪list(cost = 10^seq(-2,1, by = .5)))
summary(tune_data)
```

```
cat("Tuning shows that optimal cost is 3.162278")
```

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
     cost
 3.162278

- best performance: 0.16375

- Detailed performance results:
         cost   error dispersion
1  0.01000000 0.17375 0.04910660
2  0.03162278 0.17125 0.04678927
3  0.10000000 0.17500 0.04823265
4  0.31622777 0.17125 0.04084609
5  1.00000000 0.16875 0.03963812
6  3.16227766 0.16375 0.04185375
7 10.00000000 0.17000 0.04005205


Tuning shows that optimal cost is 3.162278
```

1.1.5 *(e) Compute the training and test error rates using this new value for cost.*

[43]:
```
tune_svm_linear<-svm(Purchase ~ ., kernel='linear', data=train,␣
 ↪cost=tune_data$best.parameters$cost)
tune_pred_train_data<-predict(tune_svm_linear, train)
table(train$Purchase, tune_pred_train_data)

cat("Train Error after tuning is:", 100 * mean(tune_pred_train_data !=␣
 ↪train$Purchase), "%\n")
```

```
   tune_pred_train_data
     CH  MM
  CH 427  60
  MM  69 244
```

```
Train Error after tuning is: 16.125 %
```

[87]:
```
tune_svm_linear<-svm(Purchase ~ ., kernel='linear', data=test,␣
 ↪cost=tune_data$best.parameters$cost)
tune_pred_test_data<-predict(tune_svm_linear, test)
table(test$Purchase, tune_pred_test_data)
```

```
cat("Test Error after tuning is:", 100 * mean(tune_pred_test_data !=␣
 ↪test$Purchase), "%\n")
```

```
   tune_pred_test_data
     CH  MM
  CH 148  18
  MM  27  77
```

```
Test Error after tuning is: 16.66667 %
```

With the use of the optimal cost, the training error has marginally decreased to 16.125%. Additionally, the test error rate decreased to 16.67% when the optimal cost was used. This, in my opinion, is probably the result of random variance in the train/test split.

1.1.6 *(f) Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the default value for gamma.*

SVM with radial kernel: without tuning

[46]:
```
set.seed(123)
svm_radial <- svm(Purchase~ ., kernel='radial', data=train )
summary(svm_radial)
```

```
Call:
svm(formula = Purchase ~ ., data = train, kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  367

 ( 181 186 )


Number of Classes:  2

Levels:
 CH MM
```

Out of the 800 available observations in the dataset, the radial basis kernel with default gamma generates 367 support vectors. Minute Maid and Citrus Hill are divided into 181 and 186 vectors, respectively.

1.1.7 train and test error

[47]:
```
pred_train_data <- predict(svm_radial, train)
table(train$Purchase, pred_train_data)

cat("Train Error radial basis kernel is:", 100 * mean(pred_train_data !=␣
 ↪train$Purchase), "%\n")
```

```
   pred_train_data
     CH  MM
  CH 446  41
  MM  70 243
```

```
Train Error radial basis kernel is: 13.875 %
```

[70]:
```
pred_test_data <- predict(svm_radial, test)
table(test$Purchase, pred_test_data)

cat("Test Error radial basis kernel is:", 100 * mean(pred_test_data !=␣
 ↪test$Purchase), "%\n")
```

```
   pred_test_data
     CH  MM
  CH 149  17
  MM  34  70
```

```
Test Error radial basis kernel is: 18.88889 %
```

SVM with radial kernel: with tuning

[67]:
```
set.seed(123)

tune_rad_data <- tune(svm, Purchase ~ ., data = train, kernel = 'radial',␣
 ↪ranges = list(cost = 10^seq(-2,1, by = .5)))
summary(tune_rad_data)

cat("Tuning shows that optimal cost for radial kernal is 1")
```

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
    1

- best performance: 0.16125

- Detailed performance results:
```

```
         cost   error dispersion
1  0.01000000 0.39125 0.04411554
2  0.03162278 0.37750 0.05027701
3  0.10000000 0.17625 0.06108112
4  0.31622777 0.17125 0.05104804
5  1.00000000 0.16125 0.04875178
6  3.16227766 0.16375 0.04226652
7 10.00000000 0.17000 0.04794383
```

Tuning shows that optimal cost for radial kernal is 3.162278

The outcomes are only marginally superior to the linear kernel. We discover that cost = 1 and the corresponding cross-validated misclassification rate of 0.16125 are the ideal values. So it appears that the radial svm is equally effective in this situation.

1.1.8 train and test error

[69]:
```
tune_svm_rad <- svm(Purchase ~ ., kernel='linear', data=train,␣
 ↪cost=tune_rad_data$best.parameters$cost)
tune_pred_train_data <- predict(tune_svm_rad, train)
table(train$Purchase, tune_pred_train_data)

cat("Train Error after tuning is:", 100 * mean(tune_pred_train_data !=␣
 ↪train$Purchase), "%\n")
```

```
   tune_pred_train_data
     CH  MM
  CH 428  59
  MM  69 244
```

Train Error after tuning is: 16 %

[84]:
```
tune_pred_test_data <- predict(tune_svm_rad, test)
table(test$Purchase, tune_pred_test_data)

cat("Test Error after tuning is:", 100 * mean(tune_pred_test_data !=␣
 ↪test$Purchase), "%\n")
```

```
   tune_pred_test_data
     CH  MM
  CH 149  17
  MM  29  75
```

Test Error after tuning is: 17.03704 %

The basic radial kernel with the lowest cost has a training error of about 16%, which is higher than the default gamma.

The basic radial kernel with the lowest cost has a test error rate of 17.03704%, which is lower than the default gamma.

1.1.9 *(g) Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Set degree = 2.*

SVM with polynomial kernal: without tuning

[74]:
```
set.seed(1212)
svm_poly<-svm(Purchase~ ., kernel='polynomial', data=train, degree=2 )
summary(svm_poly)
```

```
Call:
svm(formula = Purchase ~ ., data = train, kernel = "polynomial",
    degree = 2)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  1
     degree:  2
     coef.0:  0

Number of Support Vectors:  445

 ( 219 226 )


Number of Classes:  2

Levels:
 CH MM
```

Out of the 800 available observations in the dataset, the polynomial kernel with degree 2 generates 445 support vectors. MM and CH are divided into 226 and 219 vectors, respectively.

1.1.10 train and test error

[77]:
```
pred_train_data<-predict(svm_poly, train)
table(train$Purchase, pred_train_data)

cat("Train Error is:", 100 * mean(pred_train_data != train$Purchase), "%\n")
```

```
   pred_train_data
     CH  MM
  CH 454  33
  MM 105 208
```

```
Train Error is: 17.25 %
```

[86]:
```
pred_test_data<-predict(svm_poly, test)
table(test$Purchase, pred_test_data)

cat("Test Error is:", 100 * mean(pred_test_data != test$Purchase), "%\n")
```

```
   pred_test_data
     CH  MM
  CH 153  13
  MM  47  57
Test Error is: 22.22222 %
```

SVM with polynomial kernal: with tuning

[81]:
```
tune_poly_data<-tune(svm, Purchase ~ ., data=train, kernel='polynomial',␣
 ↪degree=2, ranges=list(cost=10^seq(-2,1, by=.5)))
summary(tune_poly_data)

cat("Tuning shows that optimal cost for radial kernal is 10")
```

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
   10

- best performance: 0.17

- Detailed performance results:
         cost   error dispersion
1  0.01000000 0.38875 0.02853482
2  0.03162278 0.36000 0.03622844
3  0.10000000 0.31500 0.02266912
4  0.31622777 0.21375 0.03884174
5  1.00000000 0.19750 0.03670453
6  3.16227766 0.18000 0.04721405
7 10.00000000 0.17000 0.04571956


Tuning shows that optimal cost for radial kernal is 10
```

1.1.11 train and test error

[83]:
```
tune_svm_poly<-svm(Purchase ~ ., kernel='polynomial', data=train, degree=2,␣
↪cost=tune_poly_data$best.parameters$cost)
tune_pred_train<-predict(tune_svm_poly, train)
table(train$Purchase, tune_pred_train)

cat("Train Error after tuning is:", 100 * mean(tune_pred_train !=␣
↪train$Purchase), "%\n")
```

```
   tune_pred_train
     CH  MM
  CH 451  36
  MM  79 234
```

```
Train Error after tuning is: 14.375 %
```

[85]:
```
tune_pred_test<-predict(tune_svm_poly, test)
table(test$Purchase, tune_pred_test)

cat("Test Error after tuning is:", 100 * mean(tune_pred_test != test$Purchase),␣
↪"%\n")
```

```
   tune_pred_test
     CH  MM
  CH 150  16
  MM  39  65
```

```
Test Error after tuning is: 20.37037 %
```

There is a decrease in error compared to the original polynomial kernel with the test error for the polynomial kernel with the most optimal cost being 20.3% and the train error being 14.3%.

1.1.12 *(h) Overall, which approach seems to give the best results on this data?*

Even though all of the models are relatively similar, the radial kernel with the default gamma performs best overall by a very slight margin. The training with this method had the lowest misclassification rate, but for test error linear model with optimal cost performed better.

2 Question 7

2.1 Fit a neural network to the Default data in the ISLR2 package. Use a single hidden layer with 10 units, and regularization. Have a look at ISLR Labs 10.9.1–10.9.2, and in class, for guidance. Compare the classification performance of yourmodel with that of logistic regression.

Loading libraries

[1]:
```
install.packages('ISLR2')
install.packages('neuralnet')
```

```r
install.packages('nnet')
install.packages('dplyr')
install.packages('keras')
install.packages('tensorflow')
library(tensorflow)
library(dplyr)
library(ISLR2)
library(neuralnet)
library(nnet)
library(keras)

use_condaenv("r-tensorflow")
```

```
The downloaded binary packages are in
/var/folders/ch/lqq67w6x5px6fcc9cnbp457m0000gn/T//RtmpCtBeDG/downloaded_packages

The downloaded binary packages are in
/var/folders/ch/lqq67w6x5px6fcc9cnbp457m0000gn/T//RtmpCtBeDG/downloaded_packages

The downloaded binary packages are in
/var/folders/ch/lqq67w6x5px6fcc9cnbp457m0000gn/T//RtmpCtBeDG/downloaded_packages

The downloaded binary packages are in
/var/folders/ch/lqq67w6x5px6fcc9cnbp457m0000gn/T//RtmpCtBeDG/downloaded_packages

The downloaded binary packages are in
/var/folders/ch/lqq67w6x5px6fcc9cnbp457m0000gn/T//RtmpCtBeDG/downloaded_packages

The downloaded binary packages are in
/var/folders/ch/lqq67w6x5px6fcc9cnbp457m0000gn/T//RtmpCtBeDG/downloaded_packages


Attaching package: 'dplyr'


The following objects are masked from 'package:stats':

    filter, lag


The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union



Attaching package: 'neuralnet'
```

```
The following object is masked from 'package:dplyr':

    compute
```

```
Error in use_condaenv("r-tensorflow"): Unable to locate conda environment␣
 ↪'r-tensorflow'.
Traceback:

1. use_condaenv("r-tensorflow")
2. stop("Unable to locate conda environment '", condaenv, "'.")
```

[2]:
```
data(Default)
```

[3]:
```
summary(Default)
```

```
 default    student       balance           income
 No :9667   No :7056   Min.   :   0.0   Min.   :  772
 Yes: 333   Yes:2944   1st Qu.: 481.7   1st Qu.:21340
                       Median : 823.6   Median :34553
                       Mean   : 835.4   Mean   :33517
                       3rd Qu.:1166.3   3rd Qu.:43808
                       Max.   :2654.3   Max.   :73554
```

split train and test data

[4]:
```
set.seed(123)
indis <- sample(1:length(Default[,1]), 2/3*length(Default[,1]), replace = FALSE)
train <- Default[indis, ]
test <- Default[-indis, ]

n <- nrow(Default)
ntest <- trunc(n/5)
testid <- sample(1:n, ntest)
y_test <- Default$default[testid] == 'Yes'
```

One thing to keep in mind is that in the example above, our default variables are No (if the Default is FALSE) and Yes (if the Default is TRUE).

Let's train our linear logistic regression model, then evaluate it using the validation set. It will forecast Default in our model. Yes, if the likelihood of default is higher than 0.5. In other words, our model will forecast P(Default|X). Our model will label X as a Default TRUE if this probability is larger than 0.5. Otherwise, it will identify X as a Default FALSE.

[5]:
```
head(train,5)
```

A data.frame: 5 × 4

	default <fct>	student <fct>	balance <dbl>	income <dbl>
2463	No	No	559.4686	61187.81
2511	No	No	905.6415	47271.35
8718	No	Yes	1806.5517	17648.20
2986	No	No	1615.2250	55219.52
1842	No	Yes	1229.4415	12158.04

2.1.1 Logistic Regression

[6]:
```
lm_model <- glm(default~student+balance+income,family="binomial",data=train)
lm_pred <- predict(lm_model, data=test, type='response') > 0.5
lm_accuracy = mean(lm_pred == y_test)

cat("Accuracy of Logistic regression model is:", 100 * lm_accuracy, "%\n")
```

```
Warning message in lm_pred == y_test:
"longer object length is not a multiple of shorter object length"
```

```
Accuracy of Logistic regression model is: 95.67957 %
```

2.1.2 Neural Network

train a neural network

First, we employ model.matrix(), which generates the same matrix as lm() (the -1 omits the intercept). This function automatically changes variables into dummy factors.

[7]:
```
x = model.matrix(default ~. -1, data=Default)
```

Using the function keras_model_sequential(), we added details about the subsequent layers in a sequential manner to the vanilla model object we generated and named modnn.

[8]:
```
x_train <- x[-testid,]
g_train <- Default$default[-testid]=='Yes'

x_test <- x[testid,]
g_test <- Default$default[testid] == 'Yes'

modnn <- keras_model_sequential() %>%
  layer_dense(units=10, activation='relu', input_shape=ncol(x)) %>%
  layer_dropout(rate=0.4) %>%
  layer_dense(units = 1, activation='sigmoid')
```

```
Loaded Tensorflow version 2.11.0
```

compile our model.

[12]:
```
modnn %>% compile(
  optimizer='rmsprop',
  loss='binary_crossentropy',
  metrics='accuracy')

# modnn %>% compile(loss = "mse",
#                   optimizer = optimizer_rmsprop(),
#                   metrics = list("mean_absolute_error")
#                  )
```

fit model to our data. Using 128 for the latter means that at each step of SGD, the algorithm randomly selects 128 training observations for the computation of the gradient

[13]:
```
history <- modnn %>% fit(
  x = x_train,
  y = g_train,
  epochs=30,
  batch_size=128)
```

predict from the final model, and evaluate its performance on the test data; neural network accuracy.

[15]:
```
nnpred <- predict(modnn, x_test) > 0.5
nn.accuracy <- mean(nnpred == g_test)
cat("Accuracy of neural network is:", 100 * nn.accuracy, "%\n")
```

```
Accuracy of neural network is: 97.3 %
```

As we can see, the accuracy of our neural network test set is approximately 97%. This is comparable to our 95% accurate linear logistic regression.

3 Question 5

3.0.1 From your collection of personal photographs, pick 10 images of animals (such as dogs, cats, birds, farm animals, etc.). If the subject does not occupy a reasonable part of the image, then crop the image. Now use a pretrained image classification CNN as in Lab ISLR 10.9.4 to predict the class of each of your images and report the probabilities for the top five predicted classes for each image.

Loading Images

[39]:
```
img_dir <- "/Users/sreeragvenugopalan/Desktop/Sem 1/Statistical Data Mining/HW/
  ↪5/animal_pics"
image_names <- list.files(img_dir)
animal_images <- length(image_names)
```

Each image must be transformed into the array "x" format required by the keras software and required by the imagenet database.

[40]:
```
x <- array(dim = c(animal_images , 224, 224, 3))
for (i in 1:animal_images) {
  img_path <- paste(img_dir , image_names[i], sep = "/")
  img <- image_load(img_path, target_size = c(224 , 224))
  x[i,,, ] <- image_to_array(img)
}
x <- imagenet_preprocess_input(x)
```

Now that the 10 photos have been imported into variable "x," we will utilize the trained network to classify them.

[41]:
```
model <- application_resnet50(weights = "imagenet")
```

After classifying our ten photos, we return the top five classes based on anticipated likelihood.

[42]:
```
model_predict <- model %>% predict(x) %>%
 imagenet_decode_predictions(top = 5)
names(model_predict) <- image_names
model_predict
```

$bear1.jpeg A data.frame: 5 × 3

class_name <chr>	class_description <chr>	score <dbl>
n02132136	brown_bear	9.983271e-01
n02133161	American_black_bear	9.804289e-04
n02504013	Indian_elephant	1.942050e-04
n02134418	sloth_bear	7.829728e-05
n01871265	tusker	6.653819e-05

$bear2.jpeg A data.frame: 5 × 3

class_name <chr>	class_description <chr>	score <dbl>
n02132136	brown_bear	9.994561e-01
n02437312	Arabian_camel	1.399826e-04
n02410509	bison	1.260052e-04
n02133161	American_black_bear	3.473239e-05
n02112137	chow	3.305958e-05

$'bull dog.jpeg' A data.frame: 5 × 3

class_name <chr>	class_description <chr>	score <dbl>
n02093428	American_Staffordshire_terrier	0.80314577
n02087394	Rhodesian_ridgeback	0.08103683
n02100583	vizsla	0.06271304
n02105412	kelpie	0.03468334
n02099849	Chesapeake_Bay_retriever	0.01001645

$cat.jpeg A data.frame: 5 × 3

class_name <chr>	class_description <chr>	score <dbl>
n02123159	tiger_cat	0.409328610
n02124075	Egyptian_cat	0.323106885
n02123045	tabby	0.249603629
n02127052	lynx	0.004285372
n04589890	window_screen	0.002157560

$cat2.jpeg A data.frame: 5 × 3

class_name <chr>	class_description <chr>	score <dbl>
n02127052	lynx	0.657612503
n02123045	tabby	0.176928356
n02124075	Egyptian_cat	0.076254085
n02123159	tiger_cat	0.072994851
n02128757	snow_leopard	0.004139737

$cow1.jpeg A data.frame: 5 × 3

class_name <chr>	class_description <chr>	score <dbl>
n02403003	ox	0.72211969
n02412080	ram	0.04176744
n02389026	sorrel	0.04012486
n02398521	hippopotamus	0.03507461
n03868242	oxcart	0.03093151

$cow2.jpeg A data.frame: 5 × 3

class_name <chr>	class_description <chr>	score <dbl>
n02403003	ox	0.49725905
n01582220	magpie	0.24009100
n02412080	ram	0.05741597
n02437616	llama	0.03806624
n02002724	black_stork	0.02035809

$lab.jpeg A data.frame: 5 × 3

class_name <chr>	class_description <chr>	score <dbl>
n02099712	Labrador_retriever	0.47187251
n02099601	golden_retriever	0.42569271
n02088466	bloodhound	0.02214054
n02109047	Great_Dane	0.01656094
n02087394	Rhodesian_ridgeback	0.01410496

$zebra.jpeg A data.frame: 5 × 3

class_name <chr>	class_description <chr>	score <dbl>
n02391049	zebra	0.9951559305
n01798484	prairie_chicken	0.0010698679
n02423022	gazelle	0.0007163025
n02422106	hartebeest	0.0003950584
n02129604	tiger	0.0003688071

$zebra2.jpeg A data.frame: 5 × 3

class_name <chr>	class_description <chr>	score <dbl>
n02391049	zebra	9.999850e-01
n02643566	lionfish	6.445944e-06
n02422699	impala	1.501251e-06
n01798484	prairie_chicken	1.404495e-06
n02423022	gazelle	9.710911e-07

When two items are identical, the image classifier has problems classifying them. For instance, the model was able to recognize a cat in image 4 but became perplexed as to what kind of cat it was, thus it discriminated it amongst three different varieties. The dog and cow picture exhibits a similar level of perplexity.