HW1, ISyE 6501

Hank Igoe

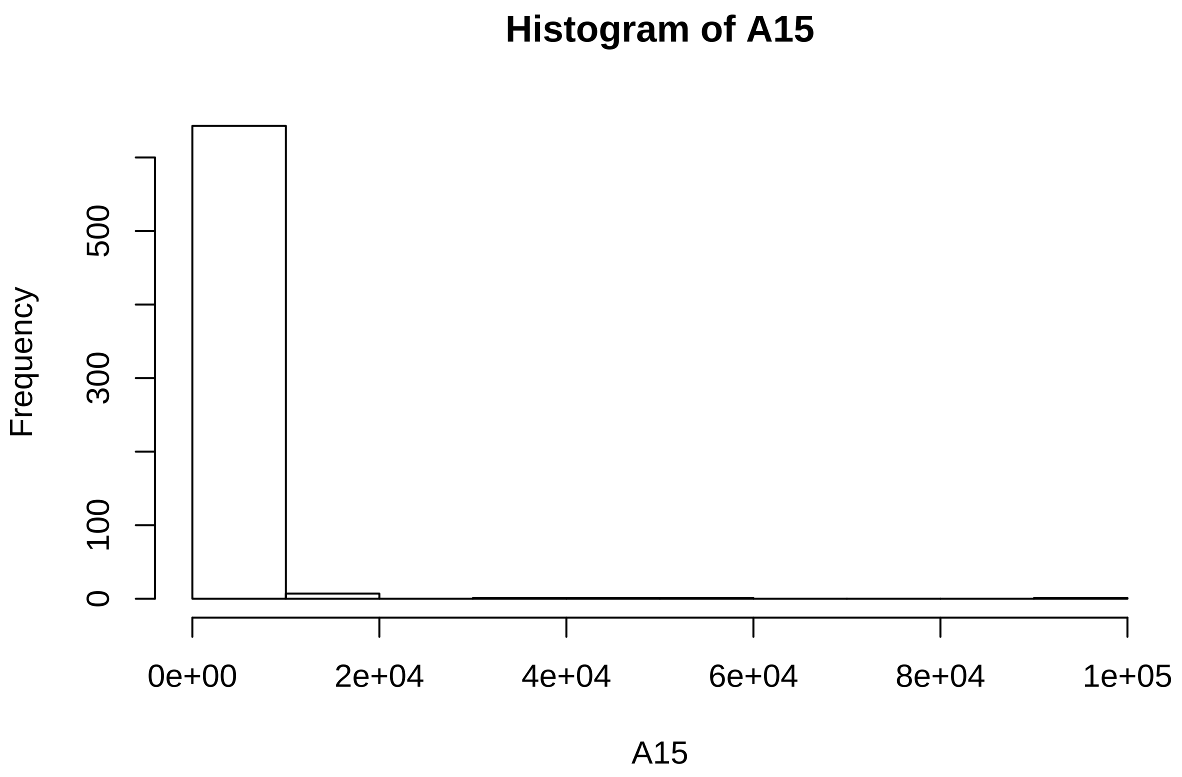
1/16/2022

Question 2.1: Real-life example.

The credit card application analysis which we undertake in the second part of the homework is an excellent example of using a classification model for its practical effects. Classification is paramount in decisions like this because it means the difference between a bank prospering due to savvy choices in granting loans to debtors who repay, or on the other hand, declaring bankruptcy as a result of granting loans to borrowers who default. The credit card data set consists of 654 vectors where each vector represents a particular applicant, and within each vector are 10 independent variables/features, 6 of which are continuous while the other 4 are binary. Because financial data is sensitive by its nature, our data set has been anonymized by replacing the column labels with generic headers: A1, A2, A3, A8, A9, A10, A11, A12, A14, A15, R1. The i in the A_i are non-contiguous since presumably some of the columns were omitted because they are irrelevant to the classification. We do know, however, that R1 is the variable which we have been tasked with classifying. The semantic meaning of R1 isn't important. Perhaps R1=1 means the loan was repaid or perhaps R1=1 means the loan defaulted. Our job is simply to develop a classification function $f(A1, A2, A3, A8, A9, A10, A11, A12, A14, A15)$ which will label (with a high probability of accuracy, of course) potential loan applicants in terms of their similarity to previous applicants. The first, fifth, sixth and ninth columns (A1, A9, A10, and A12) are binary variables while the others are continuous. The most interesting of the columns is A15. Let us load the data and take a look at a histogram and summary statistics for this column, which is the 10th column in the data frame.

```r
df <- as.matrix(read_tsv("../data/credit_card_data-headers.txt", show_col_types=F)) %>% as.data.frame
```

```r
A15 <- df[, 10]
hist(A15)
```



```r
A15 %>% summary
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##       0       0       5    1013     399  100000
```

Notice the extreme skewness present here: the mean of 1,013 is more than twice the 3rd quartile, which is just 399. That reminds me, we have yet to answer the question of which predictors we might use in our classifier. The Motley Fool lists these 7 factors[1] as being relevant to lenders when classifying loan applicants:

1. Credit score
2. Income/employment history
3. Debt-to-income ratio
4. Value of collateral
5. Liquid assets
6. Size of down payment
7. Loan term

If A15 is one of the seven criteria mentioned here, the most likely candidates are income, value of collateral, liquid assets and size of down payment. Let us get some more detail from A15. First, we will count how many of the entries are zero:

[1]https://www.fool.com/the-ascent/personal-loans/articles/7-factors-lenders-look-considering-your-loan-application/

```r
countZeros <- A15[A15 == 0] %>% length
nrows <- dim(df)[[1]]
c(countZeros, countZeros / nrows)
```

```
## [1] 275.00   0.42
```

So 275 of A15's entries are zero, representing 42% of the total. Whatever A15 represents, it must be a feature which takes the value zero frequently. Here are the counts for which A15 takes the value 1 through 5:

```r
lapply(1:5, function(i) A15[A15==i] %>% length) %>% unlist
```

```
## [1] 27  9  6  5  8
```

Thus 27 entries take the value 1, 9 entries are 2, . . . , and 8 entries have the value 5. Recall that 5 is the median, so the entries in the first half of the sorted version of A15 are small digits, the majority of which are zero. Nevertheless, its mean is 1,013 and its maximum is 100,000. At this point I do not know what to think, but I am curious to know what this column represents.

As interesting as column A15 is, however, it has taken up enough of our time. We should move on to our classification analysis.

Question 2.2.1: Optimize vanilla classifier using ksvm with type=C-svc (i.e., C classification).

The first kernel tested was the vanilla/linear kernel - "vanilladot" - and the C value that I eventually settled on was C=0.0014 which had a success rate of 0.867 = 86.7%. The success was surprisingly unperturbed by changes to C after initially setting C=100 and then testing both larger values and smaller values. At values below C=0.0014 the accuracy began to decrease. A success rate larger than 86.7% would have been preferable, but the model wasn't amenable to an increase in accuracy, which left one other aspect to optimize: the generalizability to unforeseen datasets, which is the entire goal of this endeavor, after all. Since C is inversely related to the margin parameter λ and since C=$1/\lambda$ implies that a large value of C corresponds to a small value of λ and conversely, the small value of C here implies a large value of λ, which should bode well when this model is used on test data sets. That is to say, an advantage of the small value of C is that the resulting model, namely

$$\begin{aligned}f(\mathbf{x}) &= 0.000434 * x_1 + 0.01782 * x_2 + 0.03218 * x_3 + 0.1129 * x_4 + 0.4700 * x_5 \\ &-0.2271 * x_6 + 0.1712 * x_7 - 0.0058 * x_8 - 0.02431 * x_9 + 0.0929 * x_{10} - 0.1175\end{aligned}$$

should have a similar level of accuracy on future data sets as it is not the result of over-fitting, where $\mathbf{x}$ is a feature vector, the x_j are its individual features, and x_1=A1, . . . , x_{10}=A15 (bearing in mind that the A$_i$ are non-contiguous).

Question 2.2.2: Optional exploration of non-vanilla kernels.

Out of frustration with the recalcitrance of the vanilla kernel - it simply would not budge above 86% regardless of which setting was chosen for the hyperparameter C - I tried some alternative kernels. Incidentally, the fact that 86% appeared to be an upper bound on the accuracy regardless of the setting of C led me to believe that the dataset was not linearly separable. With the exception of polydot, a polynomial kernel, the other kernels were both more accurate than vanilladot and more efficient, returning the results of the call to *ksvm* almost immediately rather than after a delay of several seconds. Here are the results of testing the RBF classifier with various values of C.

```r
# Run ksvm on df_ with C and return (a, a0, pred), i.e., (weights, intercept, predictions)
C2weightsInterceptPreds <- function(C_, kernel_, df_=df) {
  # call ksvm.  Vanilladot is a simple linear kernel.
  # kernel_ 'arg' should be one of "rbfdot", "polydot", "tanhdot", "vanilladot", "laplacedot", "besseld

  model <- ksvm(df_[,1:10],df_[,11],type="C-svc",kernel=kernel_,C=C_,scaled=T)

  # calculate a1...am
  a <- colSums(model@xmatrix[[1]] * model@coef[[1]])

  # calculate a0
  a0 <- -model@b

  # see what the model predicts
  pred <- predict(model, df_[,1:10])
  list(a, a0, pred) # return weights (a) and intercept (a0) as well
}

# Convert prediction to success rate by taking ratio:
# see what fraction of the model's predictions match the actual classification
pred2successRate <- function(pred, df_=df) { sum(pred == df_[,11]) / nrow(df_) }

Cs2RBFsuccessRates <- function(Cs, df_=df) {
  preds <-
    lapply(Cs, function(C_) C2weightsInterceptPreds(C_, "rbf", df_)[[3]])
  lapply(preds, pred2successRate)
}
```

```r
Cs2RBFsuccessRates(lapply(-3:7, function(n) 10^n), df %>% as.matrix) %>% unlist
```

```
##  [1] 0.547 0.552 0.859 0.870 0.910 0.953 0.985 0.995 0.995 0.998 1.000
```

Now we see quite a different reaction to the tuning of the kernel parameter C when the kernel used is "rbfdot," which is the radial basis function kernel[2]

$$K(\mathbf{x}, \mathbf{y}) = exp\left(-\frac{|\mathbf{x}-\mathbf{y}|^2}{2\sigma^2}\right),$$

where σ is a free parameter. When C=10^{-3}, RBF is only correct 54.7% of the time, whereas when C=10^7, RBF has 100% accuracy. The next question would naturally be whether the RBF kernel is in fact more accurate at higher settings of C than the vanilla kernel or if it is enjoying a higher level of accuracy simply by virtue of overfitting. Alas, we have yet to assemble the machinery to answer such a question, so settling that matter must wait.

Question 2.2.3: Classification using the k-NN (k-nearest neighbors) algorithm.

Now let us consider the same credit card data set, but this time using the k-nearest neighbors classification algorithm *kknn* from the R package of the same name. Since this algorithm is parameterized by k, our first task is to settle on a value of k which will yield the most accurate predictions. Experimenting with values of k ranging from 2 to 100, there was remarkably little variation in the ensuing accuracy, just like using *ksvm*

[2]https://en.wikipedia.org/wiki/Radial_basis_function_kernel

with a linear kernel. The accuracy ranged between 81% and 85%. The values of k which fared best were $k = 12$ and $k = 15$, both of which had an accuracy of 85.3%. In the interest of simplicity I settled on $k = 12$.

```r
k_optimal <- 12
responseCol <- 11 # alias for R1/target/label column

# First, a helper function which processes each row:
pred_kNN_rowi <- function(data, k_, i) {
  model_knn = kknn(as.factor(R1) ~ .,
                   data[-i,],
                   data[ i,],
                   k = k_,
                   distance = 2, # Using garden-variety euclidean distance as metric
                   kernel = "optimal",
                   scale = TRUE)
  fitted(model_knn) %>% as.integer - 1 # factors become 1, 2 instead of 0, 1
}

test_K <- function(data, k) {
  nrows <- dim(data)[[1]]
  pred_i <- function(i) pred_kNN_rowi(data, k, i)

  predictions <- lapply(1:nrows, pred_i)
  sum(predictions == data[ ,responseCol]) / nrows
}

test_K(df, k=k_optimal)
```

```
## [1] 0.853
```

In summary, we have found both a support vector model algorithm and a k-NN algorithm for this data set which are reasonably effective at classifying loan applicants based on their creditworthiness.