Introduction to Supervised Learning

Introduction

Machine learning is the art and science of giving computers the ability to learn and make decisions from data without being explicitly programmed. We describe the three branches of machine learning: Supervised learning involves analyzing labeled datasets and predicting the labels of future data points. Unsupervised learning is the art of uncovering patterns from unlabeled data. Reinforcement Learning: Agents interact with environment, learn how to optimize their behavior given a system of rewards and consequences. Draws inspiration from behavioral psychology and stochastic decision processes.

In Supervised learning we have several data points or samples called predictor variables. Our aim is to predict the target variable, given some predictor variables. The learning task is a classification task if the target variable is a discrete category and a regression if the target variable is continuous. We will often interchange between the terms feature, predictor variable, and independent variable. These terms describe the inputs into our machine learning model. The output of our model, or the thing we are trying to describe is referred as a target, response, and/or dependent variable.

For supervised learning we need labeled data for which the right output is known. The library *Scikit-learn* is a popular approach to supervised learning in python. Let us explore how supervised learning techniques are applied to real world data.

Exploratory Data Analysis

We consider the iris data set from RL Fisher included in the *sklearn* package. We will use popular packages *pandas*, *numpy*, and *matplotlib*.

```
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('ggplot')
iris = datasets.load_iris()
print(type(iris))
print(iris.data.shape)
X = iris.data
y = iris.target
df = pd.DataFrame(X, columns=iris.feature_names)
print(df.head())

   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width 
(cm)
0                5.1               3.5                1.4               
0.2
1                4.9               3.0                1.4               
0.2
2                4.7               3.2                1.3               
0.2
3                4.6               3.1                1.5               
0.2
```

```
4                    5.0                  3.6                    1.4
0.2
```

k-Nearest Neighbors

We use the algorithm k-Nearest neighbors algorithm to predict the flower species. The idea of KNN is to predict the label a of a data point by using the surrounding k data points labels.

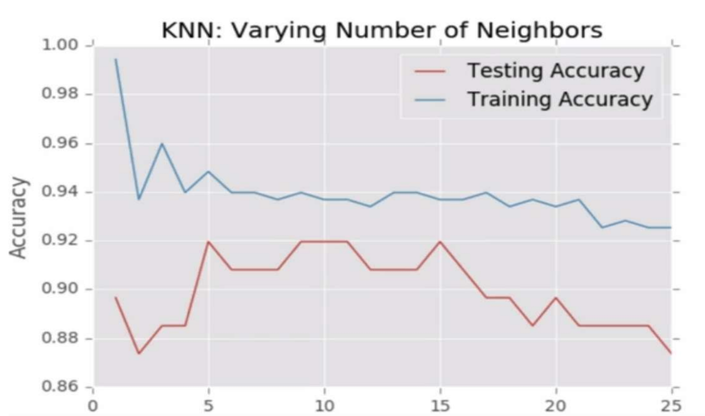
Measuring Model Performance

In classification problems accuracy is a desired metric. We view accuracy as the proportion of correct predictions. A good question is question which data should be used to compute accuracy. We should not infer the accuracy from the training data as this will not generalize to other data. Rather, we should split data into a test and validation set, where we fit and train our model on the training data and make predictions on the test or validation set. Finally, we compare predictions with known labels to measure accuracy.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,
                                                    random_state = 21,
                                                    stratify=y)
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=8)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("test set predictions: \n {}".format(y_pred))
print(knn.score(X_test, y_test))

test set predictions:
 [2 1 2 2 1 0 1 0 0 1 0 2 0 2 2 0 0 0 1 0 2 2 2 0 1 1 1 0 0 1 2 2 0 0 1 2 2
 1 1 2 1 1 0 2 1]
0.9555555555555556
```

Model Complexity:

It should be noted how the parameterization of k affects the overall model. In general models with a high k value tend to smoother, but can oversimplify the problem, while models with a small k are complex and can overfit the random effects on data. So, we aim to find a medium where we avoid overfitting, or underfitting, our data.



Supervised Learing 1: Classification

Ahern Nelson

Spring 2018

1 Introduction

We now wish to futher our study of supervised lerning by honing in on classification tasks. A classification task is one that models a relationship between a categorical target variable and feature variables. The goal of classification is to build a model capable of predicting the target label, given feature variable values, with high accuracy.

In this paper we will examine classification using Nearest Neighbors methods. Namely, we will lend significant time to the k-Nearest-Neighbors algorithm.

2 The k-Nearest-Neighbors Algorithm

Nearest-Neighbor methods use observations in the data closest to the feature variables to form neighborhoods about said points and thus estimate the target labels for new data based on these neighborhoods.

The k-nearest neighbor takes the k closest observations about some fixed observation and forms an appropriate neighborhood about the observation. We define this by:

$$\frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

Where y_i is an observation, x_i are is the feature vector, and $N_k(x)$ is the the neighborhood of x, the feature vector associated with y_i, defined by the k closest points x_i to x.

It is assumed here that the metric, $d(x, x_i)$, is defined by the usual euclidean metric. Note that this may introduce problems, or unnatural assumptions, when dealing with data containing categorical feature variables.

3 Apply nearest-neighbors in R

Here we explore the kNN algorithm in action. We consider a dataset containing data extracted from traffic sign images containing traffic sign labels and feature variables of the the RGB values.

We wish to use the kNN algorithm to predict future labels of traffic signs.

We start by reading the data into R:

```
# Read in data
signs <- read.table("signs.txt", header = T)
next_sign <- read.table("next_sign.txt", header=T)
head(signs)
```

```
##    sign_type  r1  g1  b1  r2  g2  b2  r3  g3  b3  r4  g4  b4  r5  g5  b5
## 1 pedestrian 155 228 251 135 188 101 156 227 245 145 211 228 166 233 245
## 2 pedestrian 142 217 242 166 204  44 142 217 242 147 219 242 164 228 229
## 3 pedestrian  57  54  50 187 201  68  51  51  45  59  62  65 156 171  50
## 4 pedestrian  22  35  41 171 178  26  19  27  29  19  27  29  42  37   3
## 5 pedestrian 169 179 170 231 254  27  97 107  99 123 147 152 221 236 117
## 6 pedestrian  75  67  60 131  89  53 214 144  75 156 169 190  67  50  36
##    r6  g6 b6  r7  g7 b7  r8  g8  b8  r9  g9  b9 r10 g10 b10 r11 g11 b11
## 1 212 254 52 212 254 11 188 229 117 170 216 120 211 254   3 212 254  19
## 2  84 116 17 217 254 26 155 203 128 213 253  51 217 255  21 217 255  21
## 3 254 255 36 211 226 70  78  73  64 220 234  59 254 255  51 253 255  44
## 4 217 228 19 221 235 20 181 183  73 237 234  44 251 254   2 235 243  12
## 5 205 225 80 235 254 60  90 110   9 216 236  66 229 255  12 235 254  60
## 6  37  36 42  44  42 44 192 131  73 123  74  22  36  34  37  44  42  44
##   r12 g12 b12 r13 g13 b13 r14 g14 b14 r15 g15 b15 r16 g16 b16
## 1 172 235 244 172 235 244 172 228 235 177 235 244  22  52  53
## 2 158 225 237 164 227 237 182 228 143 171 228 196 164 227 237
## 3  66  68  68  69  65  59  76  84  22  82  93  17  58  60  60
## 4  19  27  29  20  29  34  64  61   4 211 222  78  19  27  29
## 5 163 168 152 124 117  91 188 205  78 125 147  20 160 183 187
## 6 197 114  21 171 102  26 197 114  21 123  74  22 180 107  26
```

```
next_sign
```

```
##      r1  g1  b1  r2 g2 b2  r3 g3 b3  r4  g4  b4  r5  g5  b5  r6  g6  b6
## 206 204 227 220 196 59 51 202 67 59 204 227 220 236 250 234 242 252 235
##      r7  g7  b7  r8 g8 b8  r9 g9 b9 r10 g10 b10 r11 g11 b11 r12 g12 b12
## 206 205 148 131 190 50 43 179 70 57 242 229 212 190  50  43 193  51  44
##     r13 g13 b13 r14 g14 b14 r15 g15 b15 r16 g16 b16
## 206 170 197 196 190  50  43 190  47  41 165 195 196
```

We now use to KNN algorithm to predict the traffic sign type for the next_sign observation

```
# KNN lib
library(class)
sign_types <- signs$sign_type

# Classify the next sign observed
knn(train = signs[-1], test = next_sign, cl = sign_types, k=1)
```

```
## [1] stop
## Levels: pedestrian speed stop
```

Thus, we estimate that the next_sign data corresponds to a stop sign.

Now this data took an image of a traffic sign and divided it in a 4 X 4 grid for a total of 16 pixels. We then recored the RGB value for each color in each pixel.

Notice here that there is a clear distinguishment between signs types with regard to the average amount of red in 10th pixel.

```
aggregate(r10 ~ sign_type, data = signs, mean)

##    sign_type       r10
## 1 pedestrian 113.71739
## 2      speed  80.63265
## 3       stop 132.39216
```

This is how kNN identifies potential labels.

Now that we have succesfully predicted the label of one sign we will generalize to a larger test set of 59 addition road signs of the same three types.

```
test_signs <- read.table("test_signs.txt", header=T)
signs_pred <- knn(train = signs[-1], test = test_signs[-1], cl = sign_types)

signs_actual <- test_signs$sign_type
table(signs_actual, signs_pred)

##             signs_pred
## signs_actual pedestrian speed stop
##   pedestrian         19     0    0
##   speed               2    17    2
##   stop                0     0   19
```

This table gives a broad overview of the effectiveness of our model. Ideally, we want to see a diagonal matrix here, or more realistically a sparse matrix with small off diagonal values. This is a good a example of a strong application of kNN. However, notice that our algorithm only misclassifed 4 speed signs.

For a simpler metric, we tend to calculate the accuracy of our model

```
mean(signs_actual == signs_pred)

## [1] 0.9322034
```

4 Parameterizing the k in kNN

You have probably noticed that at this point we have not explicitly chosen a k in our models. In fact, all of our models up to this point have been 1-nearest-neighbor models. As stated earlier, k parameterizes the number of nearest observations about which we form our neighborhood.

Informally, we can interpret k as measure of the size of the neighborhood about an observation.

In general we realize that the smaller k is the more susceptible our model is to change and particularly overfitting. A large k value on the other hand could underfit the data and fail to encapsulate some of the more subtle details.

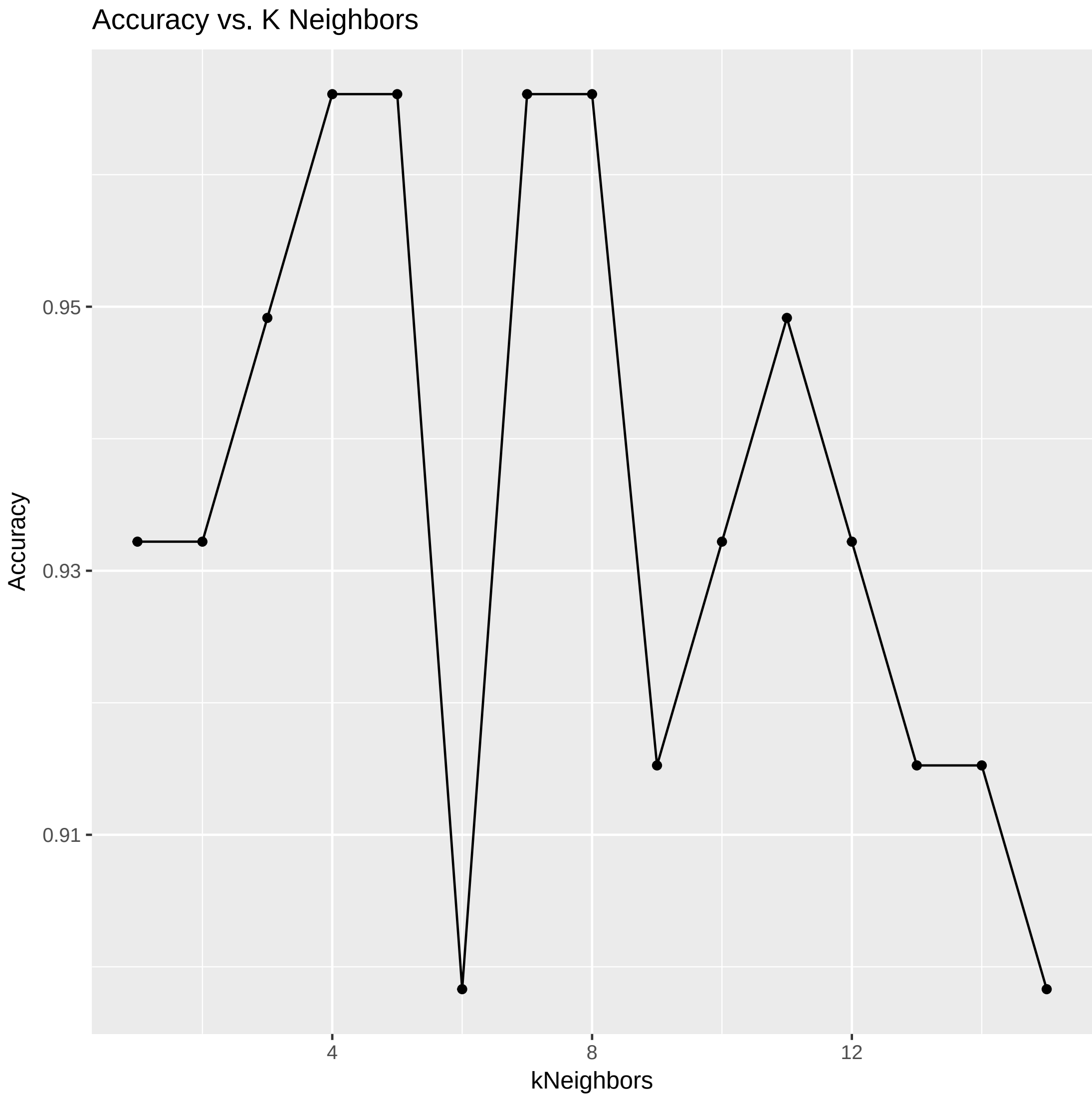
Observe how the accuracy of our previous model varies for different values of k

```
k_1 <- knn(train = signs[-1], test = test_signs[-1], cl = sign_types)
mean(signs_actual == k_1)

## [1] 0.9322034

k_7 <- knn(train = signs[-1], test = test_signs[-1], cl = sign_types, k=7)
mean(signs_actual == k_7)

## [1] 0.9661017

k_15 <- knn(train = signs[-1], test = test_signs[-1], cl = sign_types, k=15)
mean(signs_actual == k_15)

## [1] 0.8983051
```

Clearly, choosing k=7 produces a far superior model than k=1.

In general we often consider plots such as the following:

```
library(ggplot2)
accuracy <- c()
for(i in 1:15){
  k <- knn(train = signs[-1], test = test_signs[-1], cl = sign_types,
           k=i)
 accuracy <- c(accuracy,mean(signs_actual == k))
}
kNNPlot <- data.frame(Accuracy = accuracy, kNeighbors = 1:15)
ggplot(kNNPlot, aes(x=kNeighbors, y=Accuracy)) + geom_point() + geom_line() + ggtitle("Accuracy v
```



We can extract more information from the kNN algorithm by specifying 'prob = T'

```
# Use the prob parameter to get the proportion of votes for the winning class
sign_pred <- knn(train = signs[-1], test = test_signs[-1], cl = sign_types, k=7, prob= T)
sign_prob <- attr(sign_pred, "prob")

# Examine the first several predictions
head(data.frame(sign_pred,sign_prob))
```

```
##    sign_pred sign_prob
## 1 pedestrian 0.5714286
## 2 pedestrian 0.5714286
## 3 pedestrian 0.8571429
## 4       stop 0.5714286
```

```
## 5 pedestrian 0.8571429
## 6 pedestrian 0.5714286
```

This provides us with not just the predicted class label but also a probablity associated with that prediction. That is we now can interpet which predictions carry a higher certainty, and in a more detailed analysis we can explain why.

5 Conclusion

This concludes our exploration of nearest neighbor methods. This hopefully provides a glimpse of how classifaction tasks can be used in real world scenarios. The kNN algorithm itself is largely troublesome and the models used in autonomous vehicles are more complex, but the underlying motivation and concepts are the same. In the next paper we will explore bayesian methods in supervised learning and we can use this philosophy to build strong location models.

6 References

The data for this paper was obtained from the ”Supervised Learning in R” course on datacamp.com The code was written by me in guided excerises provided by datacamp.com. The presented formula for kNN was taken from the text ”Elements of Statistical Learning” by Hastie, Tibshirani, and Friedman