Supervised Learning - k-NN

Mark Asch - IMU/VLP/CSU

2023

Program

1. Data analysis
   (a) Introduction: the 4 identifiers of “big data” and “data science”
   (b) Supervised learning methods: regression, k-NN, SVM, NN, decision trees.
   (c) Unsupervised learning methods: k-means, principal component analysis, clustering.

Recall: Regression and Classification

Variables can be characterized as:

- ✔ quantitative, that take on numerical values;
- ✔ qualitative (or categorical), that take on values in one of K different classes (or categories).

The problems are then of type:

- ✔ regression when we are dealing with quantitative variables,
- ✔ classification for qualitative variables.

Examples of classification

Remark. Classification problems are more frequent than regression problems.

- Some examples :
  - ⇒ a person arrives at the emergency room with a set of symptoms that could be attributed to one of three medical conditions—which of the three is the patient suffering from?
  - ⇒ on the basis of a DNA sequence for a number of patients with and without a given disease, a biologist would like to know which mutations are dangerous and which are not?

k-Nearest Neighbors (k-NN)

✔ k-NN is a supervised learning algorithm based on similarity or proximity

✔ Utilization:

- we have a pile of objects that have been classified or labelled in some way or other
- we have similar objects that have not yet been classified or labelled

- we want to classify automatically the latter

✔ Question: is linear regression applicable here? Answer: it depends...

- LR requires an output that is a continuous variable—so, NO
- we can transform the categories into values using thresholds—so, YES

Mathematical Formulation of k-NN

For

- a given value of k,
- and a prediction point x_0,

the k-NN regression/classification is:

- identify first the k training observations that are the closest to x_0 in a neighborhood $\mathcal{N}_0$

$\Rightarrow$ then estimate $y_0 = f(x_0)$ using the average of all the responses in $\mathcal{N}_0$ in the case of a regression

$$\hat{y}_0 = \hat{f}(x_0) = \frac{1}{k} \sum_{x_i \in \mathcal{N}_0} y_i$$

$\Rightarrow$ or the majority vote in the case of a classification (categories)—compute the conditional probability of a class j as the proportion of points in $\mathcal{N}_0$ with response equal to j,

$$\Pr(Y = j | X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_i = j)$$

and classify the test observation, x_0 in the class with the highest probability.

The k-NN procedure

1. Choose a distance/similarity metric.
2. Divide the labeled dataset into training and test data (usually 80-20).
3. Choose an evaluation metric (eg. the proportion of misclassifications in the test set).
4. Execute k-NN several times, varying the value of k and checking the evaluation metric.
5. Optimize k by choosing the value that yields the best metric.
6. Create a new test set with the objects to classify and repeat.

Distance metrics

Euclidean Distance for real-valued data

Cosine Similarity between two real-valued vectors, a and b

$$\cos(a, b) = \frac{a \cdot b}{\|a\| \, \|b\|},$$

which produces a value between -1 (opposite) and 1(exactly the same) with 0 between the two, implying independence.

Jaccard Distance between two sets of objects

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|},$$

Hamming Distance between 2 character strings or 2 words, (eg. 2 DNA sequences) where we loop over the each position and check whether the letters are the same, if not we increment a counter by one. For example, the distance between «olive» et «ocean» is 4, and that between ATCGA et CTGAA is 3.

Mahlanobis Distance take into account the correlation between 2 real-valued vectors

$$d(a,b) = \sqrt{(a-b)^T S^{-1}(a-b)}$$

and is invariant with respect to scale, where S is the covariance matrix

Manhattan distance between 2 real-valued vectors

$$d(a,b) = \sum_{i=1}^{n} |a_i - b_i|$$

Remark. The choice of metric depends on the type of data... and must be made based on experience.

Advantages et Disadvantages

- Advantages:
  - ✔ simple and efficient
  - ✔ no hypothesis on the underlying distribution of data (non-parametric)
  - ✔ rapid training phase
- Disadvantages:
  - ✘ no model produced (a "lazy" method)
  - ✘ "curse of dimensionality" requiring a lot of memory when there is a large number of features
  - ✘ need to treat outliers and missing values

Training and Test Sets

- ✔ 2 phases of all machine learning algorithms:
  - ✔ a training phase where the model is created and “learns” how to treat the data
  - ✔ a test phase where we use new data to check how good the model really is

Example. Take a database with classification of individuals according to their capacity to reimburse their credit (confidence rating) . The training phase for k-NN consists simply of reading the data with points marked “high” or “low” and then classifying them as a function of income. Then, for testing, we pretend not to know the rating and we check the capacity of the algorithm to “guess” the good one. For this, we extract 20% of the data, drawn randomly from the database.

Example of k-NN - credit ratings

```
> head(data)
   age income credit
1  69     3    low
2  66    57    low
3  49    79    low
4  49    17    low
5  58    26    high
6  44    71    high
n.points <- 1000 # number of lines in the  dataset
sampling.rate <- 0.8
# we need the number of points in the test set
# to compute the misclassification rate
num.test.set.labels <- n.points * (1 - sampling.rate)
# random sample of lines for training set
training <- sample(1:n.points, sampling.rate * n.points,
               replace=FALSE)
train <- subset(data[training, ], select = c(Age, Income))
# define the training set with these lines
# the other lines go to the test set
testing <- setdiff(1:n.points, training)
# define the test set to be these other lines
test <- subset(data[testing, ], select = c(Age, Income))
cl <- data$Credit[training]
# subset of classes for training set
```

```
true.labels <- data$Credit[testing]
# subset of classes for test set,
# we retain these...
```

Choice of evaluation metric

How do we know/evaluate whether the model has done a good job? This is often complicated and context-dependent, and a domain expert should be consulted, if possible.

- ✘ minimize false negatives—eg. in cancer diagnosis
- ✘ compromise between sensitivity and specificity
  - ✘ sensitivity = the probability to correctly diagnose a sick person as sick (rate of true positives)
  - ✘ specificity = the probability to correctly diagnose a healthy person as healthy (rate of true negatives)

Definition 1. The accuracy is the ratio of the number of correct labels and the total number of labels, and the classification error is equal to one minus the accuracy.

Putting it all together...

✔ We now have:

- a measure of distance
- an evaluation metric

✔ For each individual in the test set:

- pretend not to know its label
- look at the labels of the k closest neighbors
- use the label of the majority vote to label it
- label in this way, all the members of the test set
- compute the classification error to measure our success

- All of this is performed automatically in R by a single line:

  $\Rightarrow$ `knn (train, test, cl, k=3)`

- And, in scikit-learn, by 2 lines

$\Rightarrow$ knn = sklearn.neighbors.KNeighborsClassifier(n_neighbors=3)

$\Rightarrow$ knn.fit(X_train, X_test)

The function `knn()`

- `knn()` is part of the library `class`
- the function requires 4 inputs:
  - ⇒ (1) a matrix that contains the predictors associated to the training set, `train.X`
  - ⇒ (2) a matrix that contains the predictors associated to the data for which we want to perform predictions, `test.X`
  - ⇒ (3) a vector that contains the class for the training observations
  - ⇒ (4) a value of `k`, the number of closest neighbors, to be used by the classification algorithm

The choice of k ?

We try different values, in a given range, and observe how the evaluation changes...

```
# loop and compute  misclassification rate
# for different values of k
for (k in 1:20)
{
 print(k)
 predicted.labels <- knn(train, test, cl, k)
 # use the function knn()
 num.incorrect.labels <- sum(predicted.labels != true.labels
 misclassification.rate <- num.incorrect.labels /
                          num.test.set.labels
 print(misclassification.rate)
}
```

Here is the program's output:

```
k misclassification.rate
1, 0.28
2, 0.315
3, 0.26
```

```
4, 0.255
5, 0.23
6, 0.26
7, 0.25
8, 0.25
9, 0.235
10, 0.24
```

We choose $k = 5$, which gives the lowest rate, and we apply it to the classification of an individual age 57 with a revenue of 37 000:

```
> test <- c(57,37)
> knn(train,test,cl, k = 5)
[1] low
```

Remark. We used the function `knn()` twice, with a different objective each time.

1. In the first call, the test set is a bunch of data used to evaluate the model.

2. In the second, the "test" set is in fact a new data point for which we seek a prediction. The algorithm does not

distinguish whether it is a real test, or whether we are predicting...

Modeling hypotheses by k-NN

- The k-NN algorithm is an example of a non parametric approach—there are no modeling hypotheses on the underlying probability distributions.

- But, there are other hypotheses:

  $\Rightarrow$ the notion of distance is valid in the space of data
  $\Rightarrow$ the training data are classified into at least two categories
  $\Rightarrow$ we choose the number of neighbors to use
  $\Rightarrow$ we suppose that the observed features and the labels are related

References

1. M. DeGroot, M. Schervish, *Probability and Statistics*, Addison Wesley, 2002.

2. Spiegel, Murray and Larry Stephens, *Schaum's Outline of Statistics*, 6th edition, McGraw Hill. 2017.

3. G. James, D. Witten, T. Hastie, R. Tibshirani. *An Introduction to Statistical Learning with Applications in R.* Springer. 2013.

4. Rachel Schutt and Cathy O'Neil. *Doing Data Science.* O'Reilly. 2014.