Reflection for Data Science Toolbox Assessed Coursework 2

Samantha Wise

December 21, 2018

The inference goal that we have selected was: given traffic data with non-zero duration, which model is the best for classifying between the protocol types?

Our reasoning for this is to deepen our understanding of cyber data, in particular: what is meant by 'duration'? Looking at our dataset, 97.5% of the data has zero duration. We spotted that traffic data with nonzero duration contained no 'ICMP' protocol type and there is roughly the same amount of TCP and UDP data. In the zero duration data: 58.8% of it is ICMP, 38.1% is TCP and 3% is UDP. So that pushes me to ask: why is majority of the data zero duration and why are the protocol type proportions like that? Is this typically expected or is this abnormal behaviour? There is obviously a lot of different approaches we could take, but for the sake of (as mentioned in the previous reflections) familiarising ourselves with this type of data we have chosen are inference goal as above.

We initially chose the baseline model to be the strongest out of the group's classifiers, since we wanted to see whether stacking could produce a better classifier than a standalone model. However, due to complications we agreed to set Neural Networks as the baseline model.

My chosen model was Naive Bayes [1]. Naive Bayes Classifier assumes that features are statistically independent of one another and explicitly models them as conditionally independent given the class. Whist this does appear to be applying an overly simplistic restriction on the data, in practice Naive Bayes is competitive with more complex techniques and enjoys some theoretical support for its effectiveness.

Given a data point $\vec{x} = \{x_1, \dots, x_n\}$ of n features, Naive Bayes predicts the class C_k for $\vec{x}$ according to the probability

$$p(C_k|\vec{x}) = p(C_k|x_1, \dots, x_n)\text{for } k = 1, \dots, K \tag{1}$$

Using Bayes' Theorem, this can be factored as

$$p(C_k|\vec{x}) = \frac{p(\vec{x}|C_k)p(C_k)}{p(\vec{x})} = \frac{p(x_1, \dots, x_n|C_k)p(C_k)}{p(x_1, \dots, x_n)} \tag{2}$$

Using the chain rule, the factor $(x_1, \dots, x_n|C_k)$ in the numerator can be further decomposed as

$$p(x_1, \dots, x_n|C_k) = p(x_1|x_2, \dots, x_n, C_k)p(x_2|x_3, \dots, x_n, C_k) ... p(x_{n-1}|x_n, C_k)p(x_n, C_k) \tag{3}$$

Here, the *naive* conditional independence assumption is applied. Specifically, Naive Bayes models assume that feature x_i is independent of feature x_j for $i \neq j$ given the class C_k. Using the previous decomposition, this can be formulated as

$$p(x_i|x_{i+1},\ldots,x_n|C_k) = p(x_i|C_k) \Rightarrow p(x_1,\ldots,x_n|C_k) = \prod_{i=1}^{n} p(x_I|C_k) \tag{4}$$

Hence,

$$\begin{aligned} p(C_k|x_1,\ldots,x_n) &\propto p(C_k,x_1,\ldots,x_n) \\ &\propto p(C_k)p(x_1,\ldots,x_n|C_k) \\ &\propto p(C_k)p(x_1|C_k)p(x_2|C_k)...p(x_n|C_k) \\ &\propto p(C_k)\prod_{i=1}^{n} p(x_i|C_k) \end{aligned} \tag{5}$$

Naive Bayes gives the probability of a data point $\vec{x}$ belonging to a class C_k as proportional to a simple product of $n+1$ factors (the class prior $p(C_k)$) plus n conditional feature probabilities $p(x_i|C_k)$. Since classification involves assigning the class C_k to the data point for which the value $p(C_k|\vec{x})$ is greatest, this proportional product can be used to determine the most likely class assignment.

$$p(C_a)\prod_{i=1}^{n} p(x_i|C_a) > p(C_b)\prod_{i=1}^{n} p(x_i|C_b) \Rightarrow p(C_a|x_1,\ldots,x_n) > p(C_b|x_1,\ldots,x_n) \tag{6}$$

Thus, the most likely class assignment for a data point $\vec{x}$ can be found by calculating $p(C_k)\prod_{i=1}^{n} p(x_i|C_k)$ for $k = 1,\ldots,K$ and assigning $\vec{x}$ the class C_k for which this value is largest. This is defined mathematically as:

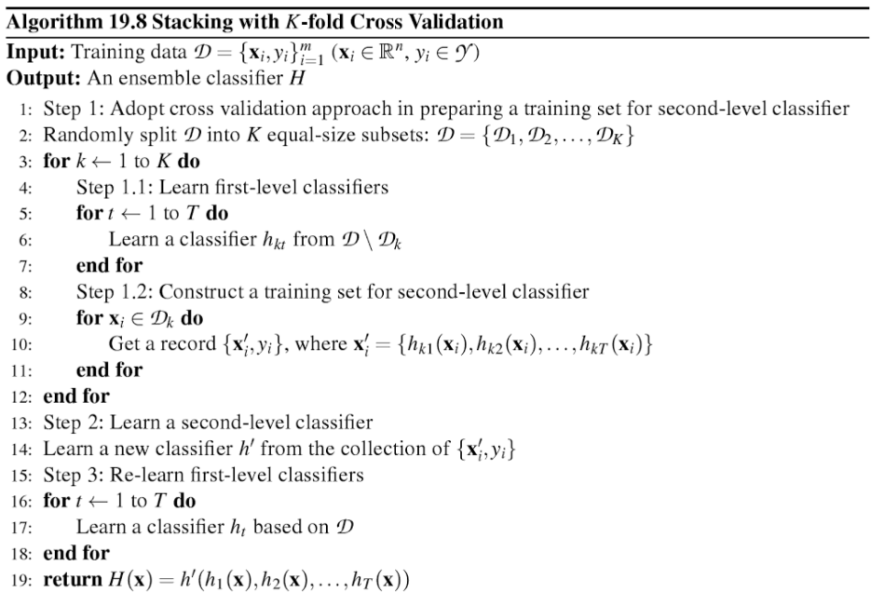
$$\hat{C} = \underset{k\in\{1,...,K\}}{\arg\max}\; p(C_k)\prod_{i=1}^{n} p(x_i|C_k) \tag{7}$$

Where $\hat{C}$ is the estimated class for $\vec{x}$ given its features $x_1, ..., x_n$

The mathematics behind stacking with K-fold cross validation is shown in Figure 1. [2]

On the previous assessment, the aim was to explore different classification methods and compare the performance of different classifier models. In this assignment, we are comparing the performance between a combined classifier and a stand alone one. This feels like an extension of the previous task. We are again enlightening ourselves with different classification models but now we are going beyond and our focus is on trying to combine the models to achieve a better model.

We encountered several setbacks in this assignment as our project proposal kept changing. I found this assignment the most challenging to complete, since coming to the end of the academic term I was having to juggle completing deadlines from other modules as well as begin my preparations for the January exams. Whilst we were able to complete the assignment in time, I felt that I spent most of the time waiting for my group members to do their part. I



```
Algorithm 19.8 Stacking with $K$-fold Cross Validation
Input: Training data $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{m}$ ($\mathbf{x}_i \in \mathbb{R}^n$, $y_i \in \mathcal{Y}$)
Output: An ensemble classifier $H$
Step 1: Adopt cross validation approach in preparing a training set for second-level classifier
Randomly split $\mathcal{D}$ into $K$ equal-size subsets: $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_K\}$
for $k \leftarrow 1$ to $K$ do
    Step 1.1: Learn first-level classifiers
    for $t \leftarrow 1$ to $T$ do
        Learn a classifier $h_{kt}$ from $\mathcal{D} \setminus \mathcal{D}_k$
    end for
    Step 1.2: Construct a training set for second-level classifier
    for $\mathbf{x}_i \in \mathcal{D}_k$ do
        Get a record $\{\mathbf{x}'_i, y_i\}$, where $\mathbf{x}'_i = \{h_{k1}(\mathbf{x}_i), h_{k2}(\mathbf{x}_i), \ldots, h_{kT}(\mathbf{x}_i)\}$
    end for
end for
Step 2: Learn a second-level classifier
Learn a new classifier $h'$ from the collection of $\{\mathbf{x}'_i, y_i\}$
Step 3: Re-learn first-level classifiers
for $t \leftarrow 1$ to $T$ do
    Learn a classifier $h_t$ based on $\mathcal{D}$
end for
return $H(\mathbf{x}) = h'(h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_T(\mathbf{x}))$
```

Figure 1: Stacking and Cross Validation

think it would be better for the group to have built the individual classification models together as we all took very different approaches (eg different packages) and as such it was difficult to perform stacking and make comparisons between the models. For example, when trying to stack LVQ and Naive Bayes model together, that approach failed because LVQ has its own prediction function 'lvqtest'.

I felt that my model (Naive Bayes) was the easiest and most straightforward to build via the 'caret package'. Sam Harding's approach to building LVQ was a little more challenging, however I managed to implement the model using 'caret'. Kishalay's model (Neural Networks) was the most complex to build and it could not be implemented via 'caret' to perform classification. I was responsible for performing the stacking, which I was able to implement again using the 'caret' package, I spent majority of the time trying to stack 'LVQ' and Naive Bayes but was unsuccessful so in order to make the stacking work in the short amount of time left, I chose the 'adaboost model'. Plotting the ROC curves was also difficult to implement and I ran into many errors, as can be seen in the markdown's commented code but I was able to compute the curves for 'Naive Bayes', 'adaboost' and the stacked model.

I will definitely take a more organised approach to the next assignment and ensure that my group spends time carefully discussing the assignment before jumping into the programming as well as setting aside an appropriate amount of time for the analysis portion.

Words: 681

References

[1] MacGonagle J. (2018). Naive Bayes Classifier. `https://brilliant.org/wiki/naive-bayes-classifier/`

[2] Aggarwal C. (2014). Data Classification: Algorithms and Applications. *Chapman and Hall/CRC*, **500**.