CS Capstone Technology Review

May 28, 2020

Reducing Patient Dose from Diagnostic Imaging Using Machine Learning

Prepared for

Oregon State University

Steve Reese

Prepared by

Group 16

Radiological Counting

Ian Brown

Abstract

Several factors are necessary to consider when making decisions on the technologies to use for each part of the project. While all Naive Bayes classifiers function by using Bayes' Theorem, there are variants that behave slightly different depending on the type of data being used. The main characteristic of each of these variants is how data with continuous values are handled. When considering the programming language that the application will be implemented in, metrics such as performance, simplicity, portability, and package selection all need to be considered. In this case, some metrics matter more than others. A choice can be made by gauging the most valuable ones and how each of the languages under consideration perform in them.

Contents

1 Naive Bayes Classifier Variants

One of the machine learning classifier types required to be implemented in this project is Naive Bayes. This is a probabilistic classifier, meaning it is relatively simple to implement and fast to execute. Several variants of Naive Bayes classifiers exist. For this project, we will consider Gaussian, Multinomial, and Complement Naive Bayes classifiers.

Bayes' Theorem, from which the family of Naive Bayes classifiers are built on, is fairly straightforward to apply when the values of each feature belong to one or more discrete classes (e.g. "red", "green", "yellow"). Since there is just one method of applying the rule to these types of features, each variant will behave roughly in the same manner if all data appears like this. However, it is quite common for models to have features that are continuous, or "real". An example of this could be a *temperature* feature whose value is a number representing degrees Farenheit. Due to the infinitely possible values for this feature, it must be accounted for in some other way in order to apply Bayes' Theorem. The main difference between each of the Naive Bayes variants is the method in which they handle (or do not handle) real-valued features.

1.1 Gaussian Naive Bayes Classifier

The Gaussian variant is usually the most common choice when values of features are continuous [1]. It works by assuming that each feature has a normal distribution. In other words, the method assumes that the feature's values appear to fit a symmetric bell curve.

1.2 Multinomial Naive Bayes Classifier

While the Gaussian classifier explicitly handles continuous values, the multinomial variant does not. Instead, values are assumed to be discrete, meaning there must be a finite number of classes in which any value of a feature belongs to. This assumption makes the multinomial variant especially well-suited for use with classifying sequences of words [2]. For example, by taking each word in a document such as an email to be a feature, the algorithm can classify the document as spam or not spam. Despite its inability to handle continuous features, the variant can still be applied using a discretization method described later.

1.3 Complement Naive Bayes Classifier

A less popular Naive Bayes variant is known as Complement Naive Bayes (CNB). Derived from the multinomial variant, CNB is especially useful when the values of features in training data sets appear to be imbalanced [2]. Further, this particular case of better performance has been proved empirically by the creators of CNB [2]. It has been noted that the CNB also often outperforms the multinomial variant in general for applications like text classification [3].

1.4 Comparison of Options

As previously stated, Naive Bayes variants differ from each other by their capacity to account for real-valued features. Therefore, the choice of variant hinges on how each of the features will be represented in the model.

1.4.1 All Discrete Features

If all of our features are discrete, any variant may be directly applied to our data set. However, the main benefit of the Gaussian variant in simplifying the distribution of continuous values will no longer be applicable, so it should not be considered for this case. In choosing between the multinomial and complement variants, the aforementioned evidence supporting the complement's tendency to outperform the multinomial seems promising. Because of this, the Complement Naive Bayes classifier will be used if features are to be represented as discreate classes.

1.4.2 At Least One Continuous Feature

If one or more features must be represented as continuous values, we could either use the Gaussian classifier directly or first discretize our features and apply either the multinomial or complement variant. This discretization would be accomplished by partitioning the data set into a number of equal-size subsets. For example, continuing from the *temperature* feature example, subsets could include <60°, 60-70°, 70-80°and >80°. While this discretization process would work, it would be detrimental to our application for a few reasons.

First, adding a discretization step will result in extra processing time needed to carry out calculations. Instead of loading sample data directly as training data, we would need an extra step of splitting up this data into each subset. The second issue is the choice of how many subsets should be used. If the value is too low, the classifier will not be accurate enough. If too few subsets are used, the final classification might be correct, but may also be unhelpful due to a large range. Alternatively, if the value is too high, the performance impact could be too large due to maintaining many data structures to classify each data entry. Because of these reasons, it would make more sense to use the Gaussian variant if features are continuous

It is the group's understanding that the spectrometer used to generate radiation counting data uses a discrete number of channels. For this reason, it can be assumed that the feature set will be discrete and therefore, the Complement Naive Bayes classifier will be chosen. If the implementation requires at least one continuous-value feature, the Gaussian variant will be chosen instead.

2 Programming Language for Implementation

One of the more fundamental decisions for this project is the choice of programming language with which to implement our application. This choice will be made based on several factors such as performance, simplicity, portability, and selection of relevant packages. Choices for this topic have been limited to Python, Java, and R.

2.1 Python

Python is currently one of the most popular programming languages in use [4]. The main draw of this language is its simple syntax and abundance of packages. This makes the language very accessible, allowing those in fields other than computer science to easily write code to accomplish tasks suited to their needs. In many cases, the same application written in other traditional programming languages could be done so in Python using a fraction of the lines of code.

This simplification comes at a cost, however. Python is an interpreted language, as opposed to a compiled language. This means that each instruction is interpreted at run time, resulting in slower execution time [5]. Since our project will need to operate in real time, performance will be important.

2.2 Java

Even more popular than Python is the Java programming language [4]. As opposed to Python and R, Java is compiled with the use of a Just-In-Time (JIT) compiler. This means that Java code is first compiled into an intermediate representation at compile time [6]. At run time, this representation is compiled further into machine code and executed [6]. The result is faster execution time compared to interpreted languages.

The inclusion of Java as a choice was done in part to the fact that the Waikato Environment for Knowledge Analysis (Weka) is written in this language [7]. If our application were to be developed in Java, this would give us the benefit of leveraging Weka's machine learning libraries in our own code [7]. Our client has stated that part of this project will be to use Weka for data analysis and visualization. Since this requirement will ensure that the team gains some degree of familiarity with Weka regardless, it may be simpler to also include it in Java code as well, resulting in less time spent researching additional packages.

2.3 R

As far as the most popular machine learning programming languages go, R has consistently ranked first for some time now [8]. Among the reasons for this is that R offers many useful built-in methods [9]. These methods continue to grow as many researchers develop and maintain them [9]. In the performance category, it is an interpreted language like Python giving it comparable results.

The main drawback to using R for this project is that the team is not as experienced in it as the other two options. Apart from this, it shares similar pros and cons to choosing Python.

2.4 Comparison of Options

The first metric that should be discussed for these language choices is performance. As previously stated, the three choices can be grouped into two categories: interpreted or compiled. Python and R are both interpreted languages while Java is compiled, giving Java the benefit of faster run time over the others.

Next, the aspect of simplicity should be considered. Java is ranked last in this category due to the fact that static typing is required and the more complicated syntax results in less readability. Python and R are somewhat similar in syntax, but Python appears more intuitive making it the winner here. The reasoning behind this is that R requires more built-in calls to create or manipulate data while Python implements data as objects which each have their own simple methods for accomplishing similar goals.

The portability of each of the possible choices is another important aspect to consider. Portability here means how easy it is to run code on different machines. Java utilizes the Java Virtual Machine (JVM) to compile bytecode so that, in theory, applications can be successfully run on any operating system that has the JVM installed [10]. Further, the Python and R interpreters are both written in the C programming language. This means that the host machine must have a proper C compiler or have pre-built binaries specific to their operating system in order to run the interpreter [10].

All languages offer a good selection of machine learning packages. Among the most popular for each language include scikit-learn for Python, Weka for Java, and e1071 for R. Due to additional popular packages such as TensorFlow and

PyTorch for deep learning classifiers, Python is preferred for this category. However, it is important to note that only our neural network implementation could leverage any deep learning packages, so this distinction is not as important.

For this project, we were given relevant code from another application. Since this code was written in Python, our client suggested that we also implement our application in Python so we can use this existing code. Also, since this project is intended to be a proof-of-concept, issues such as portability and performance are less important. Instead, the team is mostly concerned with the clarity of the code. The reasoning behind this is that if the code is simple and readable, future teams will be able to understand the general concepts easily. Then, performance-sensitive implementations could be done using the team's application as a reference. It can be concluded from these reasons that Python should be the programming language of choice for this project.

References

[1] J. Brownlee. (Apr. 10, 2016). Naive bayes for machine learning, Machine Learning Mastery, [Online]. Available: https://machinelearningmastery.com/naive-bayes-for-machine-learning/ (visited on 11/09/2019).

[2] 1.9. naive bayes — scikit-learn 0.21.3 documentation, [Online]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html (visited on 11/09/2019).

[3] J. Rennie, L. Shih, J. Teevan, and D. Karger, "Tackling the poor assumptions of naive bayes text classifiers," *Proceedings of the Twentieth International Conference on Machine Learning*, vol. 41, 2003.

[4] M. Priyadarshini. (Aug. 23, 2019). 10 most popular programming languages in 2019: Learn to code, Fossbytes, [Online]. Available: https://fossbytes.com/most-popular-programming-languages/ (visited on 11/09/2019).

[5] (Jul. 15, 2018). Why is python so slow? [Online]. Available: https://hackernoon.com/why-is-python-so-slow-e5074b6fe55b (visited on 11/09/2019).

[6] (Jun. 27, 2019). JIT in java: Understanding just-in-time compiler, Edureka, [Online]. Available: https://www.edureka.co/blog/just-in-time-compiler/ (visited on 11/09/2019).

[7] Use weka in your java code - weka wiki, [Online]. Available: https://waikato.github.io/weka-wiki/use_weka_in_your_java_code/ (visited on 11/09/2019).

[8] J. Brownlee. (May 9, 2014). Best programming language for machine learning, Machine Learning Mastery, [Online]. Available: https://machinelearningmastery.com/best-programming-language-for-machine-learning/ (visited on 11/09/2019).

[9] ——, (Jan. 14, 2016). Use r for machine learning, Machine Learning Mastery, [Online]. Available: https://machinelearningmastery.com/use-r-for-machine-learning/ (visited on 11/09/2019).

[10] Which is more portable - java or python? - quora, [Online]. Available: https://www.quora.com/Which-is-more-portable-Java-or-Python (visited on 11/09/2019).