
Final Project for CIS 419

Selecting the Optimal Shortest Path Algorithm for Road Networks

Abstract

We implemented a classifier to determine the most efficient shortest path algorithm for a given road network in the context of optimizing the response times of emergency responder units. We explore the viability of this option by training three key classifiers, including two variations on a Support Vector Machine (SVM) classifiers and a standard decision tree. In particular, we trained these classifiers on variations of Dijkstra's shortest path algorithm and the A* algorithm. We find that our ultimate choice of XXXXXXXX showed promise in most efficiently identifying an appropriate shortest path algorithm to use, such that an appropriate path is output. These classifiers draw upon principles from Machine Learning and are relevant for our target application of optimizing emergency responder times.

1. Introduction

Current research efforts in the area of shortest path algorithms have focused on specific types of graphs, and suggest cases for when a specific algorithm should be used based on characteristics of the input graph. This means today, users must either know about specific features of the graph to decide which algorithm to use, or they must choose an algorithm at random.

Often, Dijkstra's shortest path algorithm is used since it is the most general, but alternatives may outperform this choice on select input graphs. However, depending on the feature vectors specific to any given instance of a graph, another algorithm may more efficiently produce the shortest path; for the purposes of this project, we considered variations of Dijkstra's, as well as the A* shortest path algorithm.

We plan to compare the efficiencies of variations of Support Vector Machine and decision tree classifiers, trained on real-world road network data, to determine both the most

effective classifier, as well as to triangulate the most efficient shortest path algorithm for an arbitrary road network in working towards the final goal of outputting the optimal shortest path algorithm.

1.1. Motivation

Over 1,000 lives are lost every year as a result of delays in emergency response times. Fast detection of the shortest possible path between a starting point and an emergency destination could contribute significantly towards increasing the speed of emergency responders and thus, their ability to save lives. Given relevant traffic conditions at any particular point in time, the weights of the input road network could be updated to reflect current conditions and provide the emergency response team with the current shortest path.

To achieve this goal, machine learning classifier trained to detect important features of a particular road network given source and target may be able to determine and implement the most efficient shortest path algorithm. In the context of our target application, this may save valuable time that could significantly improve the response times of emergency responders.

2. Data Set

Currently, there is no one universal structure to represent graphs, since often times different applications require different internal graph structures. We chose Graph Modeling Language (GML) because it offers the following advantages: it is easy to read, understand, and parse. Furthermore, there is a lot of graph data available in this format, and many algorithms and packages have already been defined for this structure. We obtained simple GML graphs from Mark Newman's personal website (?) from which to test our parser, and eventually to test our feature extraction.

The training and test datasets are sourced from the Dryad Digital Repository under the title *Barrington-Leigh-Millard-Ball-PNAS2015-Century-of-sprawl-graphs-in-GML-format*. There were hundreds of graphs, each with thousands of nodes, all in GML format. * Add size of training and test sets.

3. Feature Generation

We used Python to preprocess our input data graphs before running them through our supervised learning algorithms. In preprocessing these graphs, we parsed the given graph data in order to obtain feature vectors that represent each graph. We used the *NetworkX* package to obtain most of the following features of our graphs and then combined all of the feature vectors into a 2D *numpy* array, where each column represented values of a particular feature and each row was a different graph. We then passed this numpy array as the X argument in the fit method of our classifier. The features we computed are given below.

1. Maximum Degree Centrality
2. Minimum Degree Centrality
3. Maximum Load Centrality
4. Minimum Load Centrality
5. Average Clustering
6. Node-to-Edge Ratio
7. Maximum Edge Length
8. Radius

We further considered Principal Component Analysis for dimensionality reduction, as the bottleneck in running time of this machine learning algorithm is in reading and choosing the features. However, given that we have only extracted a narrow range of features, this approach has limited usefulness.

4. Algorithms

4.1. Shortest Path Algorithms

For our shortest path algorithms, we chose the standard and bidirectional Dijkstra's shortest path algorithms as well as the A* algorithm. All three algorithms are contained within the *NetworkX* (NX) package, a Python package designed to handle complex networks. We wanted our three algorithms for comparison to be uniform in their implementation, so we could reliably compare the running times for training and predicting our classifiers.

4.2. Learning Algorithms

We chose to sample three different classifiers to compare their performance, given that our target application was to find the shortest geographical path between two points for emergency responders. We believe that these classifiers may be particularly relevant for our purposes because they

require only a relatively general base learning algorithm. Therefore, we know that future test data will conform to the road data we used for testing. It is important to remember that certain classifiers, such as Naive Bayes, would be unsuitable since many of the features we have computed over the graphs may be correlated with each other. Once we determined that X was the most appropriate classifier, we tuned its parameters using GridSearch over the training set based on the true labels of the training set. Having chosen this classifier, we compared the prediction results of our classifier on our training set with the true labels for the training data. This comparison enabled us to obtain evaluation metrics including accuracy, precision, and recall, against which to score our classifier and determine its viability as a solution for emergency responders.

4.3. SVM: Sigmoid and RBF Kernels

We used two types of support vector machines (SVMs) with both sigmoid and radial basis function (RBF) kernels. We chose to use support vector machines because we predict that there will be correlation between the graphs, and SVMs are capable of handling data that is not linearly separable. In the case that the final model may not be linear, an SVM with an RBF kernel is generally considered to be a default option. The sigmoid kernel, which can be used interchangeably with perceptron neural networks, has further been highly effective in practice. We predicted that the support vector machine classifiers would give us the highest accuracy of all of our classifiers.

4.4. Decision Tree

We chose to test a decision tree classifier because it can handle the graph features we use as inputs to the base learning algorithm and can split on the shortest path algorithms relatively quickly. Further, a decision tree classifier has the unique benefit of being easy-to-use and user-friendly. An emergency responder would only need to input his starting point in order to use this classifier. However, we recognize that a downside of this approach is the necessity of knowing the features of a graph in order to run this classifier. We predicted that the decision tree classifier would be the fastest of all of our classifiers.

5. Experimental Results

5.1. Error Rates

6. Algorithm Preference

7. Evaluation and Analysis

8. Conclusions