Final Project for CIS 419
Selecting the Optimal Shortest Path Algorithm for Road Networks

Abstract

We implemented a classifier to determine the fastest single-source, single-target 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 solution by training several classifiers, including two variations of the Support Vector Machine (SVM) classifier, a standard decision tree, and a boosted decision tree on variations of Djikstra's shortest path algorithm and the A$\star$ algorithm with multiple heuristics. We evaluated the accuracy, recall, precision, and prediction time to determine the best classifier, and we found that our ultimate choice of a depth-3 decision tree showed promise in most efficiently identifying the fastest 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

Past research efforts in the area of shortest path algorithms in the context of real-world road networks have focused on determining the best general shortest path algorithm. In a study by Zhan and Noon in 1998, a multitude of shortest path algorithms were evaluated on real-world road network data (**?**). The paper identified the PAPE graph growth algorithm and variations of Dijkstra as the best general algorithms. However, these results do not take into account the properties of specific graphs. 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 that works well most of the time.

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 Djikstra's, as well as the A$\star$ 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 the classifier that is most effective for an arbitrary road network at outputting the optimal shortest path algorithm. Here, we define optimality in terms of speed, accuracy, precision, and recall metrics.

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

University of Pennsylvania, CIS 419/519 Course Project.

our parser, and eventually to test our feature extraction(**?**).

The training and test datasets are extracted from the U.S. Census Bureau's Topologically Integrated Geographic Encoding and Referencing (TIGER) database, and were sourced from the Dryad Digital Repository under the title *Barrington-Leigh-Millard-Ball-PNAS2015-Century-of-sprawl-graphs-in-GML-format*. These graphs represent real-world street networks, and were output in GML format. Unfortunately, the input graphs did not have real positional data (i.e. latitude and longitude coordinates) for the nodes, so testing the $A\star$ algorithm's effectiveness became slightly less powerful, since it was impractical to implement heuristics beyond the defaults defined in the *NetworkX* package. A detailed discussion of our choice of algorithms follows in the *Algorithms* section. Initially, we ran our classifiers on a test dataset of 100 graphs and achieved a maximum training accuracy of around 75%, motivating an increase in the size of the dataset to around 200 total graphs. Memory and runtime restrictions limited the number of graphs we could use for training.

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 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 represented all the features for a given graph. Initially, we used a batch algorithm to do this feature extraction. We read in all the graphs and saved their *NetworkX* format, then ran each of the feature algorithms on each one in turn. However, we ran into memory errors while trying to do this for the entire 200-graph dataset, as most of the graphs were around 100MB in size, and so we decided to reformat our feature extraction algorithms to be online algorithms utilizing Python's Garbage Collector. Once we finished creating the feature set, we then passed this numpy array as the X argument in the fit method of each of our classifiers in turn. Ideally, we would like to have obtained more features, but the NetworkX package did not have many more useful algorithms that worked correctly on our graph dataset - many of the algorithms it provided required undirected graphs, for example. Furthermore, some of the features would have taken an infeasibly long runtime for the machines we had access to, but given additional resources, we would liked to have included features such as eigenvector centrality as well. The features we computed are given below.

1. Maximum Degree Centrality
2. Minimum Degree Centrality
3. Average Clustering
4. Edge-Node Ratio
5. Maximum Edge Length
6. Graph Radius
7. Network Density
8. Correlation Coefficient

We further considered Principal Component Analysis for dimensionality reduction on our features to speed up prediction and training time for our classifiers. However, since there is no real constraint for training time, and given that we have only extracted a narrow range of features, this approach has limited usefulness.

3.1. Label Generation

In order to determine the labels for the training set, we ran each of the shortest path algorithms we chose to compare on each graph in the training set and labeled the corresponding row in X based on which algorithm took the least time to finish. By labeling the data in this way, we made the assumption that each of the shortest path algorithms we used were correct, but since *NetworkX* is a widely used package, we decided that this was a safe assumption to make.

4. Algorithms

4.1. Shortest Path Algorithms

For our shortest path algorithms, we chose the standard and bidirectional Djikstra's shortest path algorithms as well as the A$\star$ algorithm. All three algorithms are contained within the *NetworkX* (NX) package, a Python package designed to handle complex networks. The primary reason as to why we limited ourselves to just these three algorithms is that they are the most commonly used shortest path algorithms. In addition, 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, and so we decided to avoid implementing our own versions of some algorithms (which may not have been optimized for Python, for example), and instead choose a set of algorithms that were all available from the same package. *NetworkX* provided us with this.

4.2. Learning Algorithms

We chose to sample four different classifiers to compare their performance, given that our target application was to

find the shortest path along roads 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. In order to determine the most appropriate classifier, we tuned all the different classifiers and their 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 with RBF Kernel

We used two types of support vector machines (SVMs) with radial basis function (RBF). 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 other common choice of kernel, a sigmoid kernel, which can be used interchangeably with perceptron neural networks, has further been highly effective in practice but may be less applicable for our chosen application. 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 depth-3 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 and his destination in order to use this classifier. Though, we recognize that a downside of this approach is the necessity of knowing the features of a graph in order to run this classifier, features for graphs that represent road networks in urban areas can be precomputed and provided to the responders in a format that is easy to understand. This works because the only change to the graphs would be changes in the weighting of edges, which could represent something like traffic or time to get between two nodes. As a result of this analysis, we predicted that the decision tree classifier would be the fastest of all of our classifiers. We further tested a boosted version of the decision tree to improve accuracy and precision, but this loses the advantage that it is easily human readable.

4.5. AdaBoost

We finally chose to test AdaBoost even though this approach would certainly have a longer running time than a standard decision tree. This is because AdaBoost on a decision tree is capable of boosting test accuracy without resulting in overfitting these data. Further advantages and disadvantages of this approach are similar to the discussion of decision trees above.

5. Discussion

We sampled four total variations of SVM and decision tree classifiers, and metrics for their learning accuracy precision, and recall can be found in the next section. For each classifier, we implemented GridSearch to determine the optimal parameters, which are again specified below. However, while GridSearch worked to optimize the test accuracy of each classifier, it resulted in too high a recall score in some cases, meaning that our classifier always predicted a single algorithm ($A\star$). As a result, in the case of the boosted decision tree and the SVM with a radial basis function kernel, we manually adjusted the parameters to find a classifier with the greatest consolidated accuracy, precision, and recall scores.

5.1. Error Rates

ROC curves demonstrate the ability of a classifier to effectively separate input data graphs by their respective most efficient algorithm. The higher the area under the ROC curve, the more effective the test result. Figure 1 below demonstrates the Receiver Operating Characteristic (ROC) curve given the GridSearch-chosen parameters for our SVM, decision tree, and boosted decision tree classifiers. As evident below, the SVM with an RBF kernel has the highest accuracy for GridSearch-chosen parameters.

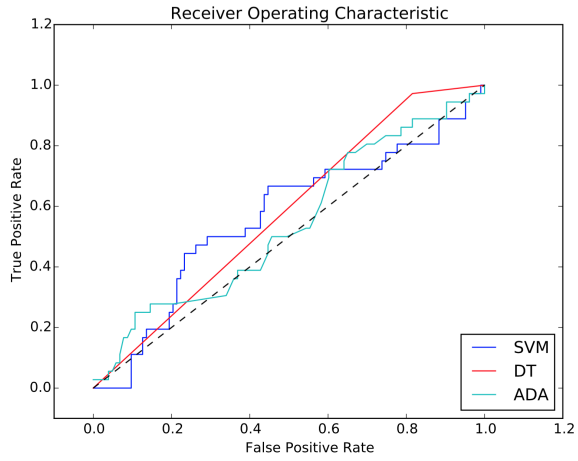
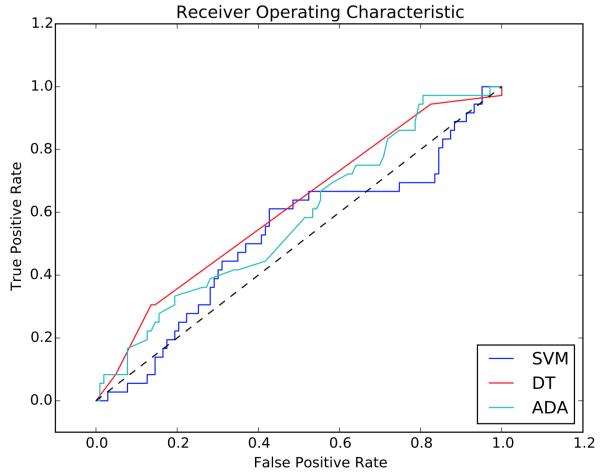


Figure 2 below represents the ROC curve given the manually optimized parameters for our SVM, decision tree, and boosted decision tree classifiers. Once these parameters were manually adjusted to reduce test recall for the AdaBoost and SVM classifiers, it is clear that the depth-3 decision tree has the highest accuracy. However, the AdaBoost and SVM classifiers with manually adjusted parameters are viewed as more effective classifiers than their GridSearch-optimized counterparts because they are more discriminating in instances, as reflected in the their lower recall scores.



The full accuracy, precision, and recall metrics for each of our classifiers are given in Tables 1-3 below, with prediction times recorded for the Support Vector Machine and decision tree classifiers.

	GridSearch SVM with 'rbf' kernel Parameters: $C = 6$, $\gamma = 0.1$
Accuracy	0.769716088328
Precision	0.756666666667
Recall	1.0
Prediction Time	0.0029308795929

Table 1: GridSearch Optimized SVM

	Manual SVM with 'rbf' kernel Parameters: $C = 500$, $\gamma = 0.1$
Accuracy	0.661870503597
Precision	0.745614035088
Recall	0.825242718447
Prediction Time	0.00171399116516

Table 2: Manually-Optimized SVM

	GridSearch AdaBoost Parameters: Rate = 0.1, $n = 50$
Accuracy	0.748
Precision	0.746
Recall	1.0

Table 3: GridSearch-Optimized AdaBoost

	Manual AdaBoost Parameters: Rate = 0.1, $n = 100$
Accuracy	0.661870503597
Precision	0.745614035088
Recall	0.825242718447

Table 4: Manually-Optimized AdaBoost

	Decision Tree Parameters: max-depth = 3
Accuracy	0.719424460432
Precision	0.780701754386
Recall	0.864077669903
Prediction Time	0.00025486946106

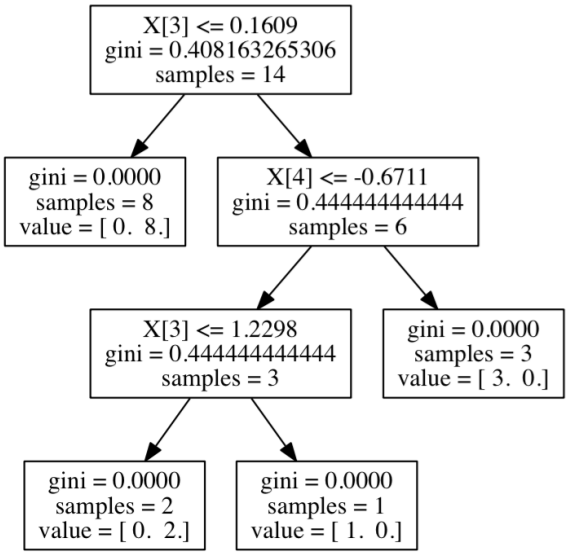
Table 5: Decision Tree

5.2. Experimental Results

Having optimized the accuracy, precision, and recall ratios of our classifiers, we found that a depth-3 decision tree had a lower prediction time than did the SVM classifier and, by definition, the AdaBoost classifier, meaning that it separated test instances more quickly than our other chosen classifiers. This classifier had a 72% accuracy and predicted a shortest path algorithm in 0.00025 seconds,

6. Algorithm Preference

Given that the depth-3 decision tree had the strongest performance in terms of both accuracy, as well as speed, of our sampled classifiers even given a range of input parameters, we used this classifier to determine algorithm preference. The visualized decision tree is given below.



7. Challenges and Future Work

A decision tree with depth 3 was shown to most accurately and most quickly predict the appropriate shortest path algorithm for a given traffic-weighted road network with an accuracy of 72% and a running time of 0.00025 seconds. However, we found that the composition of our training and test input graphs were relatively similar in composition, meaning that the classifier had the most "practice" and greatest likelihood with graphs appropriate for the standard $A\star$ algorithm. Although the size of our dataset was relatively large to ensure diversity in the input dataset, we were constrained by the running time of the preprocessing algorithms in increasing the size of the dataset. Further, we also were constrained by the information contained within our datasets. The standard .GML file format for road networks is independent of position, meaning that we were unable to determine how variations in the input heuristics of the $A\star$ algorithm might affect the accuracy, precision, and/or recall of this algorithm.

Future research in this area may benefit from a larger, more data-rich, and more diverse set of graphs to train on, perhaps enabled by greater processing power.

8. Conclusions

A decision tree with depth 3 was shown to most accurately and most quickly predict the appropriate shortest path algorithm for a given traffic-weighted road network. Using a decision tree may be especially relevant for our target end user of emergency response teams, as decision trees are highly visual and user-friendly.

References

Blackwell, T.H. and Kaufman, J.S. Response time effectiveness: Comparison of response time and survival in urban emergency medical services system. *Academic Emergency Medicine*, 9(4):288–295, 2002.

Flach, P. *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 1st edition, 2012.

Himsolt, M. Gml: A portable graph file format. pp. 1–8.

Newman, M. Network data. Technical report, Computer Science Department, University of Michigan, Ann Arbor, MI, 2013.

Zhan, F.B. and Noon, C.E. Shortest path algorithms: An evaluation using real road networks. *Transportation Science*, 32(1):65–73, 1998.