Route Recommendation based on User Preference using machine learning method

Yuxing Wang

1.Problem Description

As smartphones are widely used in daily life, guidance services have become commonplace. Route recommendation is a common application we are using every single day, especially when going for outdoor activities, going to office. There were already large amount of implementations and discussion for this algorithm in the industry. There are two major parts in the proposed algorithm:

1) Using Supported Vector Machines to build the classifier model based on user preference routes and random generated unpreferred route samples
2) Applying Monte Carlo Tree Search simulation to simulate possible routes between origin and destination, together with SVM model reward, to suggest the route recommendation for user

In the thesis, we will look into this well-known problem from other aspect, how to use user preference to provide suggestion of route recommendation. Different than other algorithms, our approach is using Monte Carlo Tree Search randomly simulate the possible routes in the map between origin and destination, it well balanced the deep and wide searching with upper confidence bound (UCB) operator, provide the near-to-best suggestion within limited time and resource. We also use support vector machines (SVM) to learn the user preferred route and randomly generated unpreferred routes to build up model, this model will be applied Monte Carlo Tree Search to evaluate the route simulation and reward the route, system finally selects the highest average route as the recommendation. We hereby attach the code implementation with Python language, unit test and few suggestions together with analysis to further improve the solution for future development.

Key Words: Individual User's Preferences router; Support Vector Machine (SVM); Monte-Carlo Tree Search (MCTS); Upper Confidence Bound (UCB)

2. Our proposal

Our proposal to solve the problem is using machine learning methodologies to:

- **Supervised learn** with user preference route data together with random unpreferred recommendation, this model is used to evaluate the reward of user preference.
- **Randomly simulate** of valid route from user input <Start, End>, and suggest the route which is preferred by user.

Support Vector Machines (**SVM**) and Monte Carlo Tree Search (**MCTS**) are introduced to algorithm as described below:

Route preference can be categorized into four aspects:

(A) Time required: Route length;
(B) Difficulty to get lost: Number of branches; Number of turns; Number of visible landmarks
(C) Safety: Route length of sidewalks; Route length of main roads
(D) Comfort: Route length of slopes

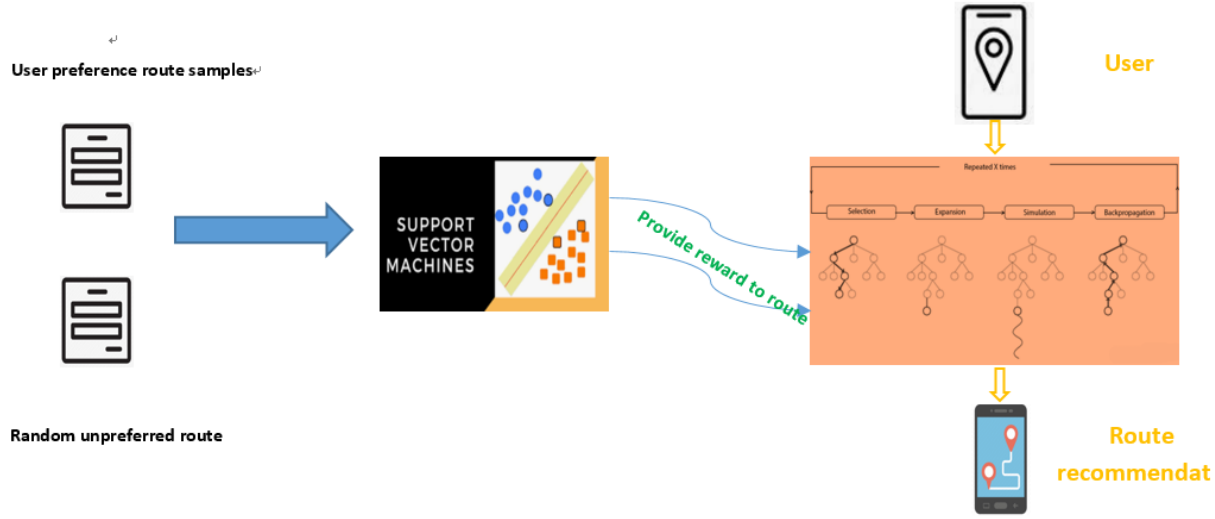


Fig.1 The route recommendation process

As the Fig.1 shown, User preference route samples are collected in user survey as positive cases (1), and algorithm generates the same number of random route as negative cases (0). **SVM** learns the feature sets from both positive and negative cases with **polynomial regression**, then build the model to evaluate the route user preference (score the **reward** for route)

MCTS is search algorithm for kinds of decision making, and widely used in playing board game, like well-known Deepmind Alpha-Go. We apply this method into route recommendation to simulate the routes.

Upper Confidence Bound (UCB) is kernel of MCTS selection action, it balances both **exploitation** (1, average SVM reward) and **exploration** (2, priority of node which has less visits)

$$UCB1 = \overline{Xj} + \sqrt{\frac{2\ln n}{nj}}$$

3. Implementation – Support Vector Machine
Dev environment: **Jupyter notebook + Anaconda**
Lib: **Sklearn.SVM**
SVM implementation:
Step 1: User survey is collected with the pairs of <start, end> point in the map, user will define their own preferred route and marked as "1", our software will generate the random route as unpreferred marked "0". Note that the number of preferred route equals to number of unpreferred route to eliminate the bias

Step 2: Features of route are extracted to generate the route table, example as Table 1. To limit

the effort, we use five features in our model evaluation, which are distance of route, whether it is sidewalk, width of road, Slope and Slope Length. User preference is also record for classifier

Step 3: Start learning using sklearn classifier with polynomial regression, the output model will be test with test dataset as well. In the MCTS, SVM model is going to reward the simulated route during simulation operation

Table 1 Route features

Route No	Distance	Sidewalk	Width	Slope	Slope length	Preference
0	1100	0	0.13	0	0.7	**1**
1	1200	0	0.10	1	0.2	**0**
2	1000	0	0.20	0	0.8	**1**
3	1800	0	0.60	1	0.3	**0**
4	1100	0	0.13	0	0.9	**1**
5	1500	0	0.10	1	0.1	**0**
6	800	0	0.20	0	0.5	**1**
7	1500	0	0.10	1	0.0	**0**
8	1100	0	0.13	0	0.5	**1**
9	1500	0	0.80	1	0.2	**0**

Code Snapshot

```
#Spliting the dataset in independent and dependent variables
X = dataset.iloc[:,:5].values
y = dataset['preference'].values
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 82)
# Feature Scaling to bring the variable in a single scale
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train1 = sc.fit_transform(X_train)
X_test1 = sc.transform(X_test)
from sklearn.svm import SVC
svcclassifier = SVC(kernel = 'poly', random_state = 0, probability=True)
svcclassifier.fit(X_train1, y_train)
y_pred = svcclassifier.predict_proba(X_test1)
```

Fig.2 Code snapshot

In this project, the code snapshot of Support Vector Machine are shown on the

4.Implementation – Monte Carlo Tree Search

In solution of route recommendation problem, we remove the expand operation from MCTS, after certain times of simulation, the node with the highest average reward will be chose as the

preferred node to start next iteration

In the extreme simple map, certain node might not have enough unique random routes to simulate, in this case, we give root node the increment, but not the selected node, in order not to artificially reduce the average reward of selected node, and make higher priority for other child node to be selected in next round of selection

The process of the use of monte carlo tree search is shown on the Fig.3 Monte Carlo Tree Search.

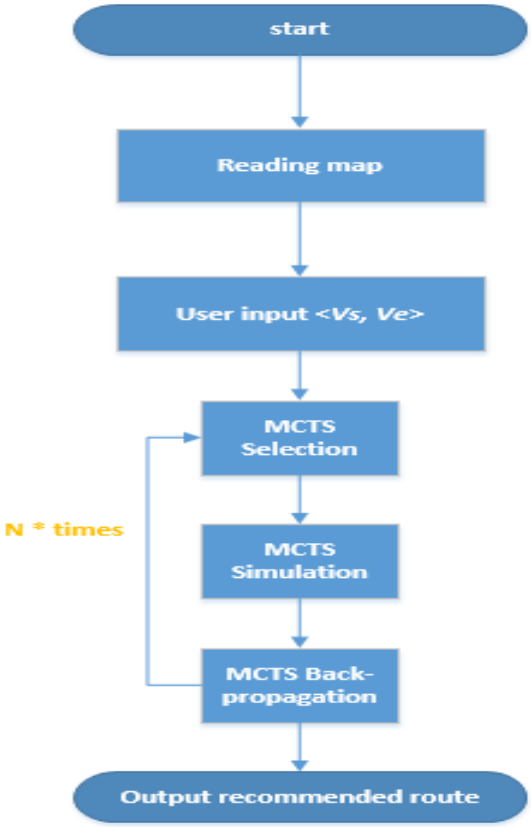


Fig.3 Monte Carlo Tree Search

```
def select(self, nowNode, simuRoute, isSelect):
    maxPUCB = 0
    maxNode = None
    for currNode, currAdge in nowNode.edges.items():
        #If current node is nowNode's parents node, don't select this node
        if currNode in simuRoute:
            for node in simuRoute:
                print(node.name)
            continue
        #
        #if the node had never been selected, the node PUCB value is infinite
        if currNode.visitN == 0 and isSelect==False:
            return currNode
        currPUCB = 0
        if currNode.visitN !=0:
            currPUCB = self.calculatePUCB(nowNode, currNode, isSelect)
            print("currPUCB:"+str(currPUCB)+" currNode:"+currNode.name)
        if currPUCB>maxPUCB:
            maxPUCB = currPUCB
            maxNode = currNode
            print("maxNode:"+maxNode.name)
    return maxNode
```

Fig.4 Code of Monte Carlo Tree Search(1)

```
def Simulation_and_Update(self, startNode, endNode):
    # generate the random route
    self.nowNode.visitN=self.nowNode.visitN+1

    allRoutes = self.allPathMap.get(self.nowNode.name+"-"+startNode.name+"-"+endNode.name)
    print("key in dict:"+self.nowNode.name+"-"+startNode.name+"-"+endNode.name)
    if allRoutes == None:
        allRoutes = self.gf.findAllRoutes(startNode.name, endNode.name, [])

        self.allPathMap[self.nowNode.name+"-"+startNode.name+"-"+endNode.name] = allRoutes
    elif len(allRoutes) == 0:
        if startNode.name != endNode.name:
            print("all routes simulated")
            mcts.selectNode.visitN=mcts.selectNode.visitN-1
            return -1
    shortestPath = []
    if len(allRoutes) != 0:
        shortestPath = self.findShortestRouteInRandom(allRoutes)
        if len(shortestPath) == 0:
            if mcts.selectNode.rewardSum!=0:
                mcts.selectNode.visitN=mcts.selectNode.visitN-1
            return -1
    elif startNode.name == endNode.name:
        shortestPath.append(startNode.name)
    shortestPathNode = []
    for nodeStr in shortestPath:
        shortestPathNode.append(self.graph.vs.get(nodeStr))
    self.simuRoute = self.simuRoute + shortestPathNode
    c_list = []
    for node in self.simuRoute:
        print(node.name)
        c_list.append(node.name)
    ar.generateFeature(c_list)
    score = ar.routeAssess()
    # svm rewards
    return score
```

Fig.5 Code of Monte Carlo Tree Search(2)

The parts of codes of Monte Carlo Tree Search are shown on the Fig.4 Code of Monte Carlo Tree Search(1) and Fig.5 Code of Monte Carlo Tree Search(2).

5. Result and Further Improvement

The Fig.6 The map is a simple map built for test the method, G<V, E> includes totally 11 nodes and 26 edges

Road **width** is visible and green triangle refers to the slope road, system learnt from user preference on page4, user likes the slope way with slightly narrow road, for sure the short distance

Our result suggests the route highlighted with **red color**

We also can see user does not care whether there is the sidewalk (in yellow) according to user preference

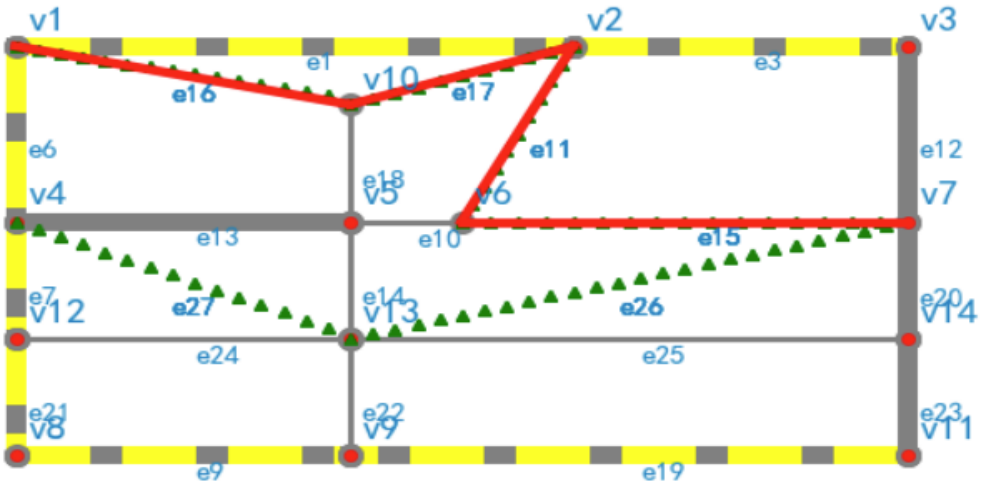


Fig.6 The map

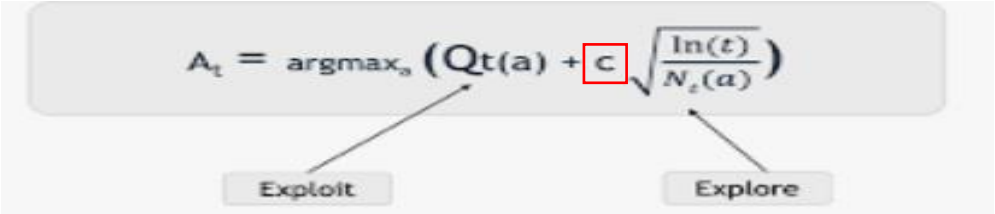
6.Further Improvement

- As simulation route is random, "**loop**" route needs to be removed, since no user will like loop route, this was implemented in code already

- User might has the different preferences for different purpose of route recommendation, for example, outdoor activities, shopping, go-to-work

- Number of simulation iteration **N** in MCTS must be big enough to evaluate high amount of routes, especially when the map complexity is high (many nodes and edges), but with limited time allowed for recommendation, N should be enable to dynamically changed (reduce), we propose to reduce the N by calculate the direct distance between the current node and end node (shorter the direct distance between current node, the smaller number of iteration)

- Similar approach like the one above, when the current node is closer and closer to the end node, **exploration** in UCB algorithm is less important than **exploitation**, constant **C** (currently square root 2) can be set to smaller value to reduce the weight of exploration



7.Conclusion

In this class and project, I have got many information about the artifical intelligence and machine learning. I am really interested in this domain and I am looking forward to learning more and putting what I have learned into practice in the future. Thanks for professor providing me this opportunity to actually know what my interestes in. What's more, at the process of doing researches, reading papers and finishing the project by myself, I get the knowledge about how to do scientific research and how to think out of the box. I have also camp up with many creativity points in this project and innovative solutions to some difficult problems. I will enter a university to pursue my master degree in machine learning . Finally, thank you, professor.