WiFi based Obstacle Detection

Rupam Kundu and Tanmoy Das

January 26, 2018

1 Abstract

Ability to move independently in space requires the knowledge of the environment which depends highly on visual perception. Vision helps to progressively develop a mental map of the environment based on obstacles and conspicuous landmarks. Hence, navigation for visually impaired/robot/drone is extremely important to adaptively plan trajectories based on environment. Exisiting infrastructure either demands expensive, sophisticated devices or fails to achive the required goal. We plan to validate whether WiFi alone can be used to find out obstacles in user's vicinity. Our initial simulations with synthetic data generation shows that combining DNN with WiFi can be promising to identify obstacles with a maximum accuracy of 70%.

2 Introduction

Obstacle detection is an indispensable component of any navigation system. Specifically, for people that are visually impaired, autonomous robots, drones, etc. Existing solution uses expensive and bulky RADAR, LIDAR and stereo cameras for obstacle detection. But these devices are directional and can only detect obstacle placed at a specific direction. The user (human or robot) needs to use manual panning (moving the camera/RADAR/LIDAR) to cover the whole $360\,\mathrm{deg}$ view. We propose a new WiFi based obstacle detection system that can detect multiple objects placed around the environment.

2.1 Setup

The setup of our problem is shown in Figure 1. The main goal of our solution is to determine the approximate directions and positions of the obstacles in the environment. To estimate the positions and directions of various obstacles, WiFi signal is leveraged. There is one antenna transmitting the WiFi signal and one or more than one antennas are for reception. The signals from these receive antennas are jointly analyzed to solve the problem of obstacle detection.

2.2 Approach

Our approach for problem formulation is presented in Figure 2. In our solution, we only try to detect the obstacles that are placed at most 15m away from the transmit antenna. This area is divided into three non-overlapping regions by three concentric circles (radius 5m, 10m, and 15m) and the transmit antenna is situated at the center of these circles.

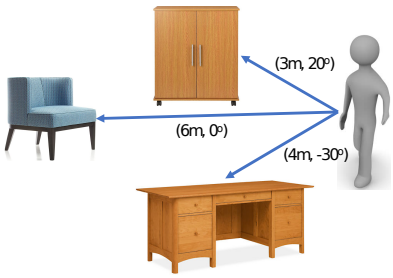


Figure 1: Problem setup.

Distance d \ Angle θ	0° - 90°	90° - 180°	180° - 270°	270° - 360°
5 m	0	1	1	1
10 m	1	0	0	0
15 m	1	0	0	1

Environment

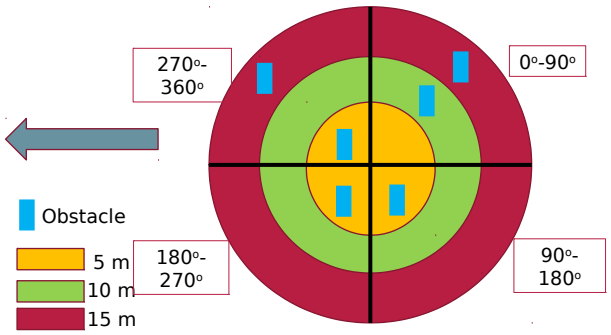


Figure 2: Approach.

Each of these regions are again divided into four disjoint sectors. As a result, the whole area inside the circle of radius 15m is divided into twelve sectors. The receive antennas are placed randomly within these area. A sector is either empty or contains one or more obstacles. Whenever a sector contains at least one obstacle, it is considered as *occupied* otherwise it is considered as *empty*.

2.3 Data Generation

As mentioned before, we are the first to use WiFi signal for omni-directional obstacle detection. To the best of our knowledge, currently there is no dataset available that can be used to train a model to solve this problem. On the other hand, it requires a huge amount of effort to create a real life labeled data set. After a lot of consideration, we have decided to use simulated data sets to train our models. The data is generated in the following way. The signal transmitted by an antenna can be modeled as

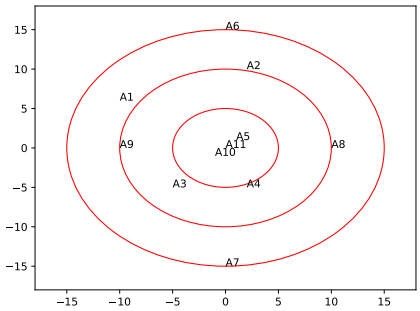


Figure 3: Antenna placements.

$$T(t) = a \exp(j2\pi ft) \tag{1}$$

Here $j = \sqrt{-1}$, a is the amplitude and f is the frequency of the transmitted signal. This signal is reflected by n obstacles in the environment and received by the antennas. The received signal to a single receive antenna can be modeled as

$$R(t) = \sum_{i=1}^{n} \frac{a}{d_i^2} \exp(j2\pi ft + \frac{2\pi d_i f}{c}) \tag{2}$$

The received signal is the sum of the reflected signals from n obstacles. The distance traveled by the signal from transmitted antenna to the i^{th} obstacle and back to the receive antennas is d_i. Due to this distance, the amplitude and the phase of the signal changes. The amplitude reduces by a factor of $\frac{1}{d_i^2}$ and an additional phase term ($\frac{2\pi d_i f}{c}$) is added to the signal. c is the speed of light.

In our data generation model, we uniformly distribute the obstacles in different sectors. The transmit antenna is placed at the center of the three concentric circles and eleven receive antennas are placed randomly as shown in Figure 3. The transmit antenna sends a signal at frequency 5.9GHz. There are multiple obstacles (1-10) placed randomly in the 12 sectors and the signal at a receive antenna is calculated using the Equation 2. We divide the collected data into three different settings. In the first setting, the received data from three antennas (A1, A2, and A3) are used. In the second setting, received data from six antennas (A1, A2, A3, A4, A5, and A6) are used. In the last setting, data from all the eleven antennas are used.

3 Background

Our work for obstacle detection using WiFi signal is related to the works in the fields of WiFi based localization, indoor navigation, depth sensing using RADAR/LIDAR and camera based navigation.

WiFi based localization: WiFi based localization [**?**, **?**, **?**, **?**] has gained a lot of attention in recent years due to the ubiquity of the WiFi signal. This localization schemes can detect the position of a WiFi transmitter (Smartphone, WiFi AP) by analyzing the receive signal using MUSIC algorithm [**?**]. However, the number of detected objects are limited by the number of antennas at the receiver. As the number of antennas on a single WiFi based device is limited due to space constraints, these schemes are not suitable for obstacle detection in an environment containing a large number of objects.

Indoor navigation: Indoor navigation is important for the guidance of blind people, autonomous robot and drones. As the GPS signal is not available, some GPS like guidance service is required in indoor environment. Indoor navigation using WiFi [**?**, **?**, **?**, **?**] has been explored in the recent past. This type of services build a map of the indoor environment using the WiFi signal strength. This map is leveraged to estimate the position of a WiFi receiver. But they are unable to detect the presence and the position of the obstacles within the environment.

WiFi Imaging: WiFi signal is used for creating the image of an object placed directly in front of the antennas [**?**]. Using the technique proposed, it is possible to detect the shape of a single object but it is not possible to detect multiple objects in the environments.

RADAR/LIDAR: RADAR/LIDAR uses radio/light waves to measure the Timeof-Flight as the signals bounce back from nearby obstacles. However, due to the high directionality of RADAR/LIDAR, either a mechanical movement is required or an array of such sensors need to be employed resulting in a bulky design. Moreover, commercially available RADARS and LIDARS consumes high power (nearly 8 watts).

Camera based navigation: Single camera based SLAM techniques lack scale. Commercially available long range stereo cameras consume high power (nearly 4 watts) [4], while the small ones [7] have very short range (2.5meters).

4 Models

There are many different models for classification that could be suitable for our problem. But due to lack of time, we only get the chance to use the following three models.

4.1 Logistic Regression

Mathematically, the logistic regression can be defined as a model that best estimates the parameter β such that:

$$Y = \begin{cases} 1, if \;\; \beta_0 + \beta_1 x > 1 \\ 0 \end{cases} \tag{3}$$

```
 Optimizing for Sector: 0
|------------------------------------------------------------------------------|
|------------------------------------------------------------------------------|
|-----------------------------Logistic Regression-----------------------------|
|------------------------------------------------------------------------------|
()
# Tuning hyper-parameters for accuracy
Best parameters set found on development set:
{'penalty': 'l2', 'C': 0.1, 'max_iter': 1000, 'solver': 'liblinear'}

Grid scores for all parameters:
0.599 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'True', 'C': 0.001, 'max_iter': 1000, 'solver': 'lbfgs'}
0.599 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'False', 'C': 0.001, 'max_iter': 1000, 'solver': 'lbfgs'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'True', 'C': 0.01, 'max_iter': 1000, 'solver': 'lbfgs'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'False', 'C': 0.01, 'max_iter': 1000, 'solver': 'lbfgs'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'True', 'C': 0.1, 'max_iter': 1000, 'solver': 'lbfgs'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'False', 'C': 0.1, 'max_iter': 1000, 'solver': 'lbfgs'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'True', 'C': 1, 'max_iter': 1000, 'solver': 'lbfgs'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'False', 'C': 1, 'max_iter': 1000, 'solver': 'lbfgs'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'True', 'C': 10, 'max_iter': 1000, 'solver': 'lbfgs'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'False', 'C': 10, 'max_iter': 1000, 'solver': 'lbfgs'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'True', 'C': 100, 'max_iter': 1000, 'solver': 'lbfgs'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'False', 'C': 100, 'max_iter': 1000, 'solver': 'lbfgs'}
0.599 (+/-0.001) for {'penalty': 'l2', 'C': 0.001, 'max_iter': 1000, 'solver': 'liblinear'}
0.608 (+/-0.001) for {'penalty': 'l2', 'C': 0.01, 'max_iter': 1000, 'solver': 'liblinear'}
0.608 (+/-0.001) for {'penalty': 'l2', 'C': 0.1, 'max_iter': 1000, 'solver': 'liblinear'}
0.608 (+/-0.001) for {'penalty': 'l2', 'C': 1, 'max_iter': 1000, 'solver': 'liblinear'}
0.608 (+/-0.001) for {'penalty': 'l2', 'C': 10, 'max_iter': 1000, 'solver': 'liblinear'}
0.608 (+/-0.001) for {'penalty': 'l2', 'C': 100, 'max_iter': 1000, 'solver': 'liblinear'}
0.599 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'True', 'C': 0.001, 'max_iter': 1000, 'solver': 'sag'}
0.599 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'False', 'C': 0.001, 'max_iter': 1000, 'solver': 'sag'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'True', 'C': 0.01, 'max_iter': 1000, 'solver': 'sag'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'False', 'C': 0.01, 'max_iter': 1000, 'solver': 'sag'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'True', 'C': 0.1, 'max_iter': 1000, 'solver': 'sag'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'False', 'C': 0.1, 'max_iter': 1000, 'solver': 'sag'}
0.608 (+/-0.002) for {'penalty': 'l2', 'warm_start': 'True', 'C': 1, 'max_iter': 1000, 'solver': 'sag'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'False', 'C': 1, 'max_iter': 1000, 'solver': 'sag'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'True', 'C': 10, 'max_iter': 1000, 'solver': 'sag'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'False', 'C': 10, 'max_iter': 1000, 'solver': 'sag'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'True', 'C': 100, 'max_iter': 1000, 'solver': 'sag'}
0.608 (+/-0.001) for {'penalty': 'l2', 'warm_start': 'False', 'C': 100, 'max_iter': 1000, 'solver': 'sag'}
```

Figure 4: Grid Search for Logistic Regression Parameters.

It is different from Linear Regression where Y can be continuous. Here, the output Y is logistically defined by a condition, hence *catagorical*. The parameters that are tunable to train a *Logistic Regression* classifier are mainly: **1.** Solver for Optimization **2.** C (Penalty parameter C of the error term) **3.** Penalty (l1 or l2 norm) **4.** Warm Start (reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution) **5.** Measurement Metric(Accuracy,Precision,Recall etc) **6.** Maximum Iteration.

4.2 Linear SVM

Given a training dataset of n points of the form $(x_i, y_i), \forall\, i \epsilon [1, n]$ where, each x_i is a M dimensional vector and each y_i is either 0 or 1, *Linear SVM* tries to find the *maximum-margin-hyperplane* that seperates the group of points $X, X \epsilon [x_1, x_2, ..., x_n]$ for which $y_i = 0$ and the group of points $Y, Y \epsilon [x_1, x_2, ..., x_n]$ where $y_i = 1$.

The parameters that are tunable to train a *Linear SVM* classifier are mainly: **1.** Loss **2.** C (Penalty parameter C of the error term) **3.** Penalty (l1 or l2 norm) **6.** Measurement Metric(Accuracy,Precision,Recall etc) **6.** Maximum Iteration. However, currently the SVM library from *tf.contrib.learn.LinearClassifier* offers the flexibility to vary the *optimizer* [*tf.train.FtrlOptimize*, *tf.contrib.linear_optimizer.SDCAOptimizer*] and also the corresponding [*learning_rate*, *l1_regularization_strength*]. We have used the SVM from *Tensorflow* library. **Note:** We implemented *Kernel SVM* too. But currently it is taking too much time in our machines due to which we could not generate the results. The performance on a shrinked dataset is very poor. So to maintain fairness we excluded Kernel SVM for comparison.

4.3 Neural Network

Neural Network (Deep Learning) is a framework consisiting of highly interconnected elements (a.k.a. neurons) that is capable of processing information analogous to human brain. Recently, after the advent of high performance computational devices (CUDA processors), *Deep Neural Network* has a profound impact in the

research community. The *Tensorflow* and *Keras* platforms offers vast flexibility to run DNN frameworks.

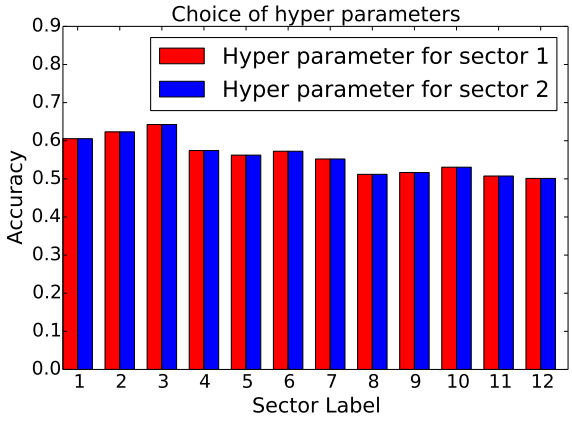
The parameters that are tunable to train a *DNN* are mainly: **1.** Epochs **2.** Batch Size **3.** Number of Layers **4.** Number of Neurons in each Layer **5.** Optimizers (SGD, ADAM, ADAGRAD etc) **6.** Learning Rate for optimizers **7.** Activation function ('tanh', 'sigmoid','softmax')

5 Experiments

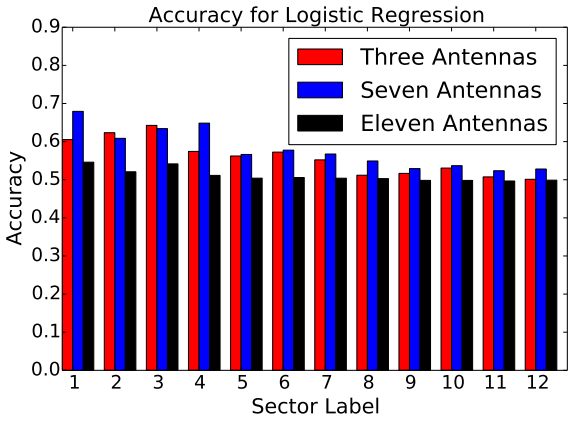
We try to find a classifier that determines if a sector is occupied or empty. In our model, we have a classifier for each sector. Logistic regressions, Linear SVM and DNN are the three models we studied. Each of these models are tested on three different settings 3 receive antennas, 7 receive antennas, 11 receive antennas the positions of which is summarized in Section. The input dataset records the phase and amplitude of the superimposed signals obtained in each antennas. Keras [6], Tensorflow[7], Scikit-learn[8] are different library frameworks used for creating the models. The codes are written in python and ran on Machines with varied configuration [GPU: GeForce GTX 1050 4GB memory, GeForce GTX 1080 12GB memory, CPU: Intel Core i7].

5.1 Logistic Regression

We have used *sklearn.linear_model.LogisticRegression*. Also to search for the best set of parameters we have used *sklearn.model_selection.GridSearchCV*. Firstly, we optimized the parameters independently for Sector 0 and Sector 1 and evaluate the accuracy for other sectors with the same set of parameters respectively. We have used GridSearch to find among the following set:



(a) Evaluating accuracy for all sectors for best params optimized w.r.t. Sector 0 and Sector 1.



(b) Accuracy for Logistic Regression.

Figure 5: Illustrates the performance of DNN

1. Solver: 'lbfgs[1]', 'liblinear[2]', 'sag[3]'
2. C: $[1e-3, 1e-2, 1e-1, 1, 10, 100]$
3. Warm_start: ['True','False']

[1]Limited-memory BFGS
[2]Library for Large Linear Classification
[3]Stochastic Gradient Descent

4. max_iter: [1000]

5. penalty':['l2']

A snapshot of such run is shown in Figure 4. The best parameters we got are: penalty: 'l2', warm_start: 'False', 'C': 100, max_iter: 1000, 'solver': 'sag'.Figure 5(a) infers that the accuracy does not change significantly opitimizing for one sector and running the same parameters for other sectors. Subsequently, we have used the parameters obtained for Sector 0 to evaluate for different Settings. Finally, Figure 5(b) shows the performance for all the sectors for three different number of receive antennas. The result concludes that increasing the antennas from 3 to 7 increases the accuracy but it decreases for 11. So an interesting observation in this case is that, the optimum number of antennas is between 3 and 7. However, we believe the position of antennas are accountable for low accuracy rate. We have randomly arranged the antennas. Hence, some antennas might be closer to objects in one sector which helps the receiver to pick up stronger signals unlike some picking up low power signals. Hence, the stronger signals are likely to create predominance when normalized to the same scale. The classifier fails to identify patterns in the signals corresponding to those sectors leading to a low accuracy rate.

5.2 SVM

We have used *tensorflow.contrib.learn.LinearClassifier*. We have also exhaustively searched for the best possible parameters. The one that we obtain is *optimizer = tf.train.FtrlOptimize(learning_rate =0.1,l1_regularization _strength =0.1)*. Figure 6 shows the performance for all the sectors for three different number of receive antennas. Likewise in previous case, increasing the antennas from 3 to 7 increases the accuracy but it decreases for 11. Here also, we believe the position of antennas are accountable for low accuracy rate.

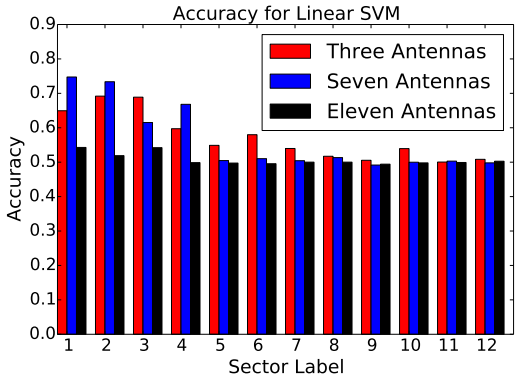


Figure 6: Accuracy for Linear SVM.

5.3 Neural Network

We have used Keras framework to run our DNN network. The DNN model is summarized as follows:

1. Input Layer, Activation: "Relu[4]"

2. 5 Dense Layer: 500 neurons, Activation: "Relu" , followed by Dropout: 0.2

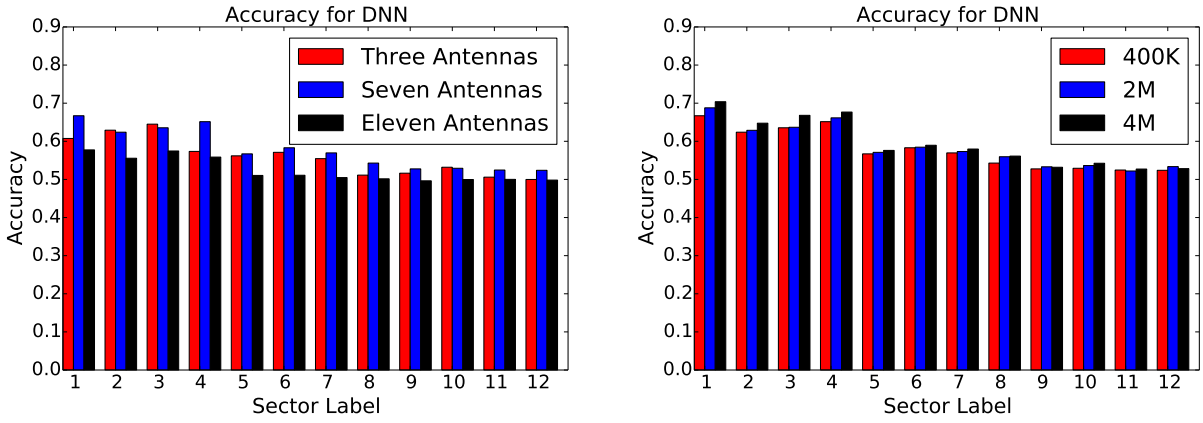
3. Output Layer: Num_classes, Activation: "Sigmoid [5]"

To check the best parameters we exhaustively searched for the same and the best parameters we got are: **1.** Optimizer: "Adamax(lr=0.001, beta_1= 0.9, beta_2 =0.999, epsilon= 1e-08, decay = 0), **2.** Batch Size=500 **3.** Epoch=10. Figue 7(a) shows the performance for all the sectors for three different number of receive antennas. Likewise in previous cases, increasing the antennas from 3 to 7 increases the accuracy but

[4]Relu or Rectified Linear Unit is an activation function: $f(x) = 1 + exp(x)$

[5]Require catagorical output in our case

it decreases for 11. Here also, we believe the position of antennas are accountable for low accuracy rate. Howver, the performance of the sectors beyond 5 are not changing as such using all the tools mentioned above. So we checked the length of training set affects the performance of those sectors or not. So we increased the dataset from 400K to 2M and 4M and ran DNN with the same parameters mentioned above. Figure 7(b) shows the performance for different datasizes. The accuracy of the sectors improves increasing the datasets still leaving room for further improvement and optimization.



(a) Accuracy for DNN for different antennas. (b) Accuracy for DNN for different data sizes.

Figure 7: Illustrates the performance of DNN

6 Discussion

Figure 8 shows the comparison of three models for 7 antenna settings. The average accuracy for all sectors is shown for each model. DNN performance is little bit better. However, overall performance affected due to non-uniform performance across all sectors. Summarizing, we observe the following inference from the evaluations. The hyper parameters selected for one classifier (associated with a sector) works well for other classifiers too. Performance improves as the number of receive antennas increase (from 3 to 7). As the number of receive antennas increases the performance improves then it starts to degrade (beyond 7 antennas). Hence the optimal antennas for our design is between 3 and 11. Optimized antenna placement study is required for enforcing uniform accuracy across all sectors. Optimized antenna together with bigger datasets still leaves room for further improvement and optimization.

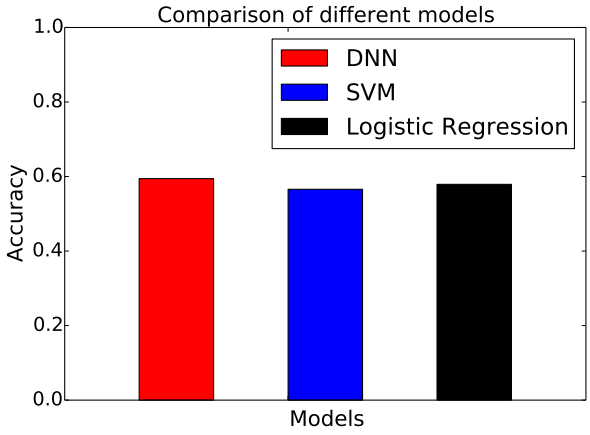


Figure 8: Model comparison.