COMP6208 Advanced Machine Learning

Student Mini-Projects

Alexander Ally (aa2g11)
Hendrik Appel (hja1g11)
Samir Moussa (sm28g11)

School of Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Thursday 7th May 2015

Abstract

This is a report describing the preprocessing methods, machine learning algorithms and validation techniques used to find a solution to a typical real-world machine learning problem in the driver telematics analysis space. The task is to model individual drivers based on their driving behaviour specified by their trip data to identify whether a particular trip has been driven by a driver or otherwise. Several approaches were studied and utilised to find a practical solution to a challenging task and our experiments are detailed in this report. We found that ensemble tree-based methods and deep belief networks performed fairly well at classifying trips.

1 Introduction

The task detailed in this report is to use telematic data to identify driver signatures. This problem was initially released as a challenge available to members of the machine learning community provided by Kaggle[1] and supported by multinational insurance company AXA. For automobile insurers, understanding the driving behaviour (telematics) of their customers allows them to quantify liability and price their insurance policies according to the risk associated with different drivers. For drivers, policies can be better suited to their experience and driving mannerisms. The end task is to develop an algorithmic signature or "telematic fingerprint" of a driver to identify when a particular trip has been driven by that driver.

Deciding to solve this machine learning problem was due to the appeal of facing challenges and complexities involved in dealing with real-world information and being able to apply machine learning methods to extract useful information and make sense of it. As demonstrated throughout the report, obstacles were far from few.

1.1 Dataset

The AXA dataset contains over 2,700 drivers and each driver has 200 trips associated. Each trip is specified by a variable-length list of (x, y) coordinates describing the position of the driver at each second relative to the starting position. To protect the privacy of drivers and their information, all trips are centred to start at the origin $(0, 0)$, randomly rotated and partially truncated from either the beginning or end of the trip. A typical set of 200 trips for a driver are plotted in figure 1. False trips are assigned to each driver and but it is not possible to distinguish between true positive and true negative trips due to the absence of labels. The size of the entire dataset is just over half a terabyte with exactly 547,200 trips to predict.

1.2 Challenges

As expected with most real-world data, there were a variety of challenges that needed to be overcome in order to start experimentation. The challenges faced, specifically with this problem,

[1] kaggle.com/c/axa-driver-telematics-analysis

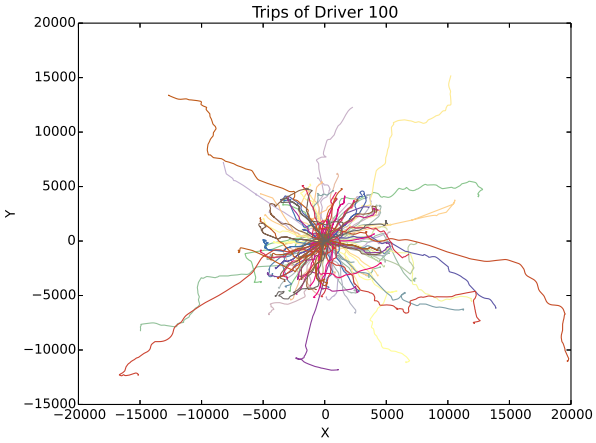

Figure 1: Typical illustration of trips for a driver.

was the unavailability of labels, presenting us with an unsupervised learning problem. Our models therefore needed to consider learning latent variables or take on clustering methods to try and find some structure to the data. Moreover, the fact that coordinates do not offer any useful description, some preprocessing and feature extraction was required. The features may be extracted manually using prior knowledge of the problem domain or learned though hidden variables as in neural networks.

Since the dataset is very large, it was expected to take a fairly long time to perform preprocessing and learn a model. To overcome this, some parallelisation work was required, utilising the university's available computing power. Finally, the data included some degree of noise and missing values due to the inaccuracies associated with GPS readings. Another challenge is that the dataset gathered was unlabelled. This meant that any validation performed by the team would be inexact, however an exact accuracy could be made by submitting to kaggle.

2 Analysis & Experimentation

2.1 Preprocessing

Forming useful features from unstructured data allows learning algorithms to perform well while minimising the amount of redundant data. A system named *representation learning* requires a model to not only learn a mapping function from input to output, but intrinsically learn a representation of the data within the model. As discussed later, unsupervised learning techniques, particularly deep learning, aim to

apply representation learning to establish a expressive representation of the underlying data, enough to extract useful information and reason about new information.

Since the raw data is unstructured and does not provide useful, or in fact any description of the important underlying features surrounding that of a driver and their driving behaviour, much of the complications faced throughout the project arose from the extraction of effective features. The trips were merely composed of sequences of GPS coordinates taken at one-second intervals. The data had to be processed and features were to be designed before machine learning techniques could be applied to make predictions. In addition the GPS coordinates given in the data were subject to inaccuracy.

2.2 Data Smoothing

Because the features were often based upon first and second order gradients of the original data, noise caused a lot of problems. Particularly when the speed of the vehicle was low, small fluctuations in the position of the vehicle caused a small velocity and acceleration to be reported. When directional information is extracted from these gradients (and magnitude information thrown away) there would appear to be wild changes in direction when really the vehicle was stationary and there was small fluctuations in the position information being reported.

In order to solve this problem a number of different avenues were explored. One simple approach was report any velocity with a magnitude of less than a small value to be set to 0. However this ignored smoothness issues that occurred at higher speeds. To remedy this a simple filter was constructed by applying the Discrete Fourier Transform and reconstructing th data using fewer components. This does appear to work well in general but in sections where the driver is crawling along at a very low speed were the filter wrongly assumes that the driver is stationary. This can be observed in the plot in figure 2.

Another approach attempted was to apply the Savitzky-Golay filter to smooth the data. Savitzky-Golay smooths data by convolution, fitting a polynomial to successive windows over the data points using a least squares optimisation. One can see that in figure 3 that the problem of threshold smoothing is does not appear and that

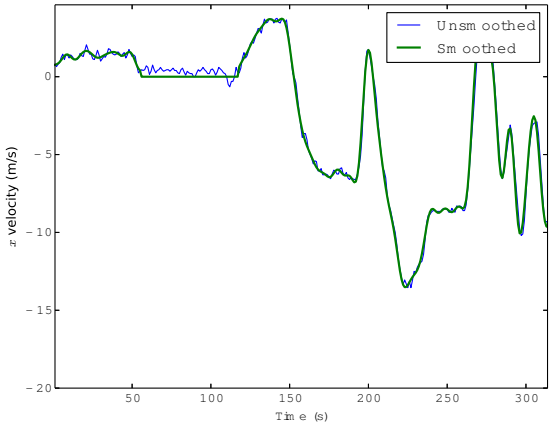

Figure 2: Fourier smoothed velocity component.

the data appears to be much smoother. However some jaggedness remains in periods of low velocity.

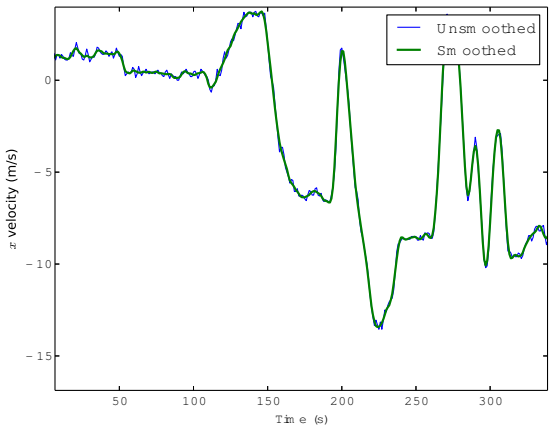

Figure 3: Savitzky-Golay smoothed velocity component.

2.3 Feature Extraction

As explained previously, the data does not contain anything that resembles a set of features. So in order to be able to perform machine learning techniques features were first designed and extracted.

Histogram Features

A number of different histograms could be taken over speeds, accelerations, acceleration gradient and angular change of the driver. It was hoped that different driving styles would be reflected by differing distributions in these computed measurements. One could also take histograms over combinations of measurements. For example a feature used was a histogram of speed multiplied

by angular change. It was hoped that would encapsulate a driver's behaviour whilst he or she made turns. Another feature was a histogram over the cosine of angular change multiplied by speed. Since the cosine of an angle is 1 at zero it was hoped that this histogram would encapsulate a driver's behaviour during straighter sections of road. Additionally histograms were taken when using longer time periods, particularly with angular change which was typically very small over the one second intervals.

Corner Features

Other features explored were those associated with corners. It was theorised that the number of corners may characterise the type of route a driver would take. For example a driver that prefers to take back roads may encounter more corners than a driver who prefers main roads. Before such features could be extracted the corners had to first be identified. Algorithm 1 shows the procedure used for identifying corners. C is a set of corners which is iteratively added to. Vector $\mathbf{v^{(i)}}$ refers to the *ith* velocity in a trip. The algorithm cycles through the velocity vectors keeping a cumulative sum of distance. If the cumulative distance (denoted by d_{curr}) exceeds a certain value (denoted by d_{thresh}) or the angle between the initial velocity and the current velocity is no longer increasing, a check is then made to see if the angle between the current velocity and the initial velocity (denoted by c_{curr}) is above some threshold (denoted by c_{thresh}). If c_{curr} is above the threshold, c_{thresh}, then corner has been identified and the process is repeated from after the corner. Otherwise the process is repeated again from one place in front of the start of the current search.

A check is also made to ensure that the vehicle is travelling at a reasonable speed otherwise random fluctuations (even with smoothing) cause problems for the algorithm.

Figure 4 demonstrates the effectiveness of the algorithm 1 by showing a trip with the corners found by the procedure highlighted in red.

Algorithm 1 Identifying corners

```
C ← {}
while i < m do
    d ← ||v^(i)||_2
    if d < 1 then
        continue
    end if
    j ← i + 1
    d_curr ← d
    c_prev ← 0
    loop
        c_curr ← angle(v^(i), v^(j))
        if d_curr > d_thresh or c_curr < c_prev then
            if c_curr > c_thresh then
                C ← C ∪ {(i, j)}
                i ← j
            end if
            break
        end if
        c_prev ← c_curr
        j ← j + 1
    end loop
    i ← i + 1
end while
```

Straights

Similarly to corners, features were created regarding slightly longer and straight sections of road. The aim was to try and isolate a driver's driving style on straight sections, specifically hoping to capture their driving characteristics on motorways. To do this straight sections had to be identified. Straights were defined as sections of movement where the difference in distance travelled and the euclidean distance was below some threshold. The threshold was made relative to the distance travelled as the magnitude of the difference would vary with the length of the straight. A minimum length was also set for straights.

Algorithm 2 defines how straights are identified. The algorithm makes use of the vector **v** containing coordinates and the vector **dists** which contains the euclidean distances between the points. There are two loops, one for the start index and another for the end index, the total distance travelled between these two indices is calculated. The euclidean distance between the start and end section is calculated by subtracting the two position vectors and taking the norm. The absolute difference of these two values is then compared to the threshold. The threshold is the maximum percentage of the distance travelled that the straight can vary by, this allows longer slightly curved roads to be counted. If the straight is within the threshold and the above the minimum length then the straight is added to a map indexed

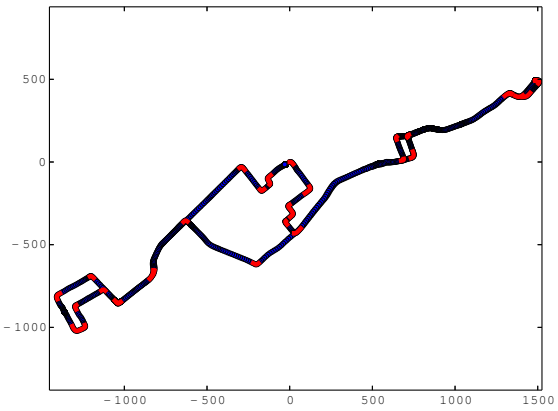

Figure 4: Plot showing trip with corners highlighted in red.

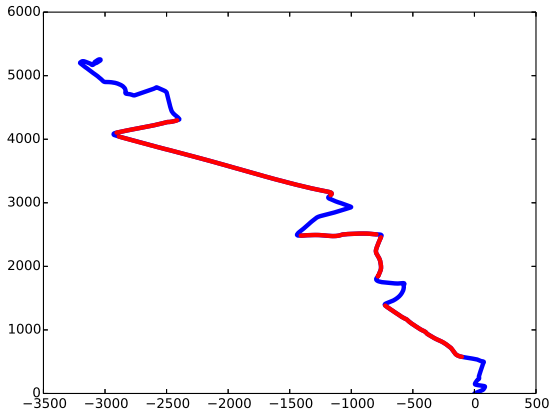

Figure 5: Plot showing a trip with straights identified in red.

by the start index and storing the end index along with the length of the straight. Figure 5 shows an example of the straight sections that are identified with this algorithm for a specific route.

Features extracted from this data include data about the length of the straights and the speeds at which the user drove along them.

Algorithm 2 Identifying straights

$S \leftarrow \{\}$
$i_{\text{end}} \leftarrow 0$
while $i < length(\textbf{dists})$ **do**
  if $i \leq i_{\text{end}}$ **then**
    continue
  end if
  for $j \leftarrow i+1, length(dists)$ **do**
    $dist_{\text{sum}} \leftarrow sum(\textbf{dists}^{(i:j)})$
    $dist_{\text{diff}} \leftarrow dist_{\text{sum}} - ||\mathbf{v}^{(i)} - \mathbf{v}^{j+1}||_2$
    $below \leftarrow abs(dist_{\text{diff}}) < t * dist_{\text{sum}}$
    if $dist_{\text{sum}} > min_{length}$ **and** $below$ **then**
      $S[i] = (j+1, dist_sum)$
      $i_{end} = j$
    end if
  end for
  $i \leftarrow i+1$
end while

Distance, Velocity & Time-Related Features

A subset of the full set of features (Appendix X) corresponds to distance, speed and time related attributes of trips. A randomly sampled trip from the dataset is plotted in figure 6 with coloured points indicating areas of stationary periods (black) and low (yellow) and high (orange) velocities. The thresholds determining "high" and "low" velocities were manually configured to best represent the natural motion of a driver along a trip.

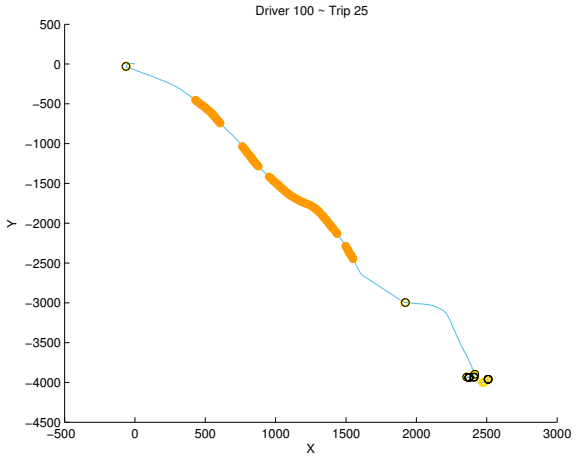

Figure 6: Points when the driver is stationary (black) and with low (yellow) and high (orange) velocities.

2.4 Machine Learning

Whilst the problem scenario prescribes an unsupervised learning technique one can also leverage the fact that the data is labelled. And although the problem is such that some of the data points are mislabelled, supervised learning techniques can still be applied in the hope that the models learned will be general enough such that the mislabelled examples don't affect predictions much. As such a number of different supervised and unsupervised approaches were used in an attempt to solve the problem.

Unsupervised Learning Approach

The unsupervised approach taken was a form of outlier detection wth each driver being considered separately. Firstly dimensionality reduction is performed through principle components analysis. Then the mean is computed and the Mahalanobis distance is computed from each data point to the mean. If it is assumed that the dataset is distributed according to a Gaussian distribution, one can infer that the distances to the mean are distributed according to a χ^2 distribution where the degrees of freedom is equal to the number of features. One can then use the cumulative density function $F(x)$ to compute $1 - F(x)$. This is the probability that a another data point will be further away from the mean than it. If this value is greater than 0.5 then the trip is classified as belonging to the driver otherwise it is classified as not belonging to the driver.

Supervised Learning Approach

For each driver a machine is trained on 180 of their trips labelled as 1 and 180 trips selected from 18 other drivers selected at random each labelled as 0. This process is repeated 10 times such that all trips for that driver have a prediction made on them.

This technique also afforded the use of rudimentary validation. One could test the trained classifiers on the data. And although this would not give an exactly correct score. It was good enough to be able to test different approaches before getting the correct score from a submission to kaggle.

Gradient Boosting Trees

Gradient boosting is a technique for combining many weak learning machines into a single machine which achieves better performance. First outlined by Freidman [1] this proved to be one of the more successful techniques for making predictions based upon the data. The particular machine used by this team was provided by the scikit-learn library and 100 boosting stages were used.

Random Forests

Random Forests are another ensemble learning machines. They train many different decision trees on different parts of the data. These trees are then averaged. The motivation behind this technique is that decision trees often have low bias but high variance meaning they often overfit the data. It is hoped that the averaging reduces the variance with only a low cost to bias.

Support Vector Machines

The Support Vector Machine (SVM) is a maximal margin classifier first outlined by Vapnik and Cortes[2]. It also allows classification of problems which aren't linearly separable through use of kernel functions. The implementation used was the scikit-learn implementation which is based on libsvm.

To use SVMs the data was first reduced in dimensionality through use of principle components analysis to 20 dimensions.

This was used as an alternative to deep learning and tree based ensemble methods to see how it compares for this particular problem domain.

Deep Learning

While many of the common approaches have demonstrated sufficient performance tackling, machine learning, statistical and prediction-based problems, more recent but advanced methods in the field of *deep learning* have attracted much attention due to their vastly clever architectures and training algorithms. The field started to gain traction when Geoff Hinton [3] released a paper on the discovery of a new training algorithm for deep belief networks.

Deep learning architectures aim to model the structure of the mammal brain, organised in multiple, deep levels [4]. In the context of image perception, the primary visual cortex serves to extract local features from scenes or images in the form of bars, edges or lines, then build higher level encodings that compute more precise details about the objects being observed [5]. Deep learning tackles the problem of learning representations using this neurologically inspired mechanism by providing a distributed description of its input in a hierarchical structure of progressively more abstract representations.

A typical deep learning model takes the form of a conventional feedforward neural network with many hidden layers and a deep architecture. However, the term 'deep' does not signify the property of a model having many layers but rather the training algorithms used to learn a model in the manner of stacking visible and hidden layers. Depending on the type of architecture, models

may be *over-complete* (more hidden nodes than visible nodes), or *under-complete* (more visible nodes than hidden nodes). It has been shown that over-complete representations with appropriate regularisation lead to better performing models [6].

To assess the performance of deep learning on our problem, a *Deep Belief Network* (DBN) was trained with the features extracted at the preprocessing stage. A DBN comprises a series of stacked *Restricted Boltzmann Machines* (RBMs). An RBM is a probabilistic graphical model of connected nodes, or more specifically, an energy-based model of variables (or nodes) $\mathbf{x}$ where the energy of the graph is of the form $p(\mathbf{x}) = e^{-\mathbf{E}(\mathbf{x})}$, where $\mathbf{E}$ is an energy function. An RBM has two layers – a visible layer and hidden layer. The objective is to minimise the energy of the model with respect to the weights between the layers. The training algorithm used is called (one-step) *contrastive divergence* [7] and works by sampling between the layers so that the hidden layer learns the visible layer at a different dimension while capturing essential information. After training an RBM, another can be stacked on top by using the previous hidden layer as the visible layer for the next RBM.

To build a DBN, three RBMs were trained an 'unfolded' into a feedforward neural network with one output node representing the binary decision value of whether a trip belongs to the driver. Each driver was represented by a model specified by the parameters in tables 1 and 2.

Table 1: DBN Parameter Settings

Parameter	**Value**
Activation function	Sigmoid
Batch size	50
Momentum	0
Alpha	1
Training epochs	5
Model epochs	50
No. Workers	16

Table 2: DBN Layers

Layer	**No. nodes**
Input	N_F
Hidden	250
Hidden	250
Hidden	250
Output	1

2.5 Validation

The score given to a classifier is the area under curve (AUC) of the ROC curve. This provides a score of between one and zero. One being a perfect score. Predicting all positives would result in a score of 0.5.

2.6 Large Scale Implementation

Processing large amounts of data presented many issues and we found the frequency of feedback hindered by the scale of the dataset.

Since ten models needed to be trained per driver, for the supervised learning approach, the computational expense of running the supervised models was very high. Especially since there are in excess of 2700 drivers. In an attempt to remedy this the team used the university's computing structure which included machines with 64 CPU cores. Through use of Python's multiprocessing module. This allowed the task to be parallelized and shared amongst many cores, massively increasing the speed at which the models could be run.

3 Results

3.1 Random Forests

Random forests provided good results and a relatively quick training time. The plot shown in figure 7 shows how the number of estimators used in the random forest affected the score. One can observe that increasing the number of estimators generally increased the AUC score of the classifier although these returns appeared to dwindle after the number of estimators reached around 120.

3.2 Gradient Boosting Classifier

The gradient boosting classifiers resulted in good AUC scores, providing roughly comparable prediction performance to Random Forests. The plot shown in figure 8 shows how the learning rate affects prediction performance. It shows that increasing the learning rate slightly increases the performance until the learning rate is 0.5 before the score starts to decrease.

3.3 Support Vector Machines

SVMs performed fairly poorly on this particular problem. Table 3 shows how SVMs performed

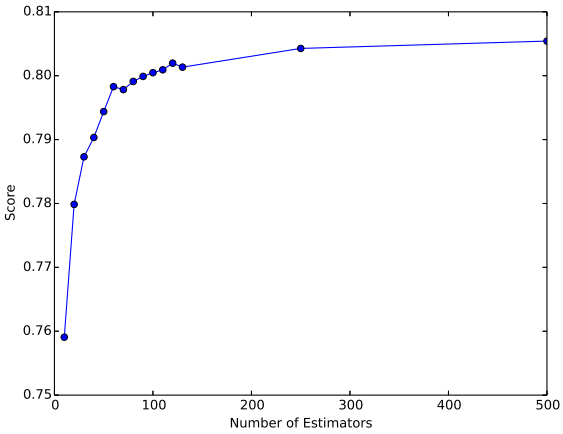

Figure 7: Plot showing the number of estimators in a random forest against AUC score

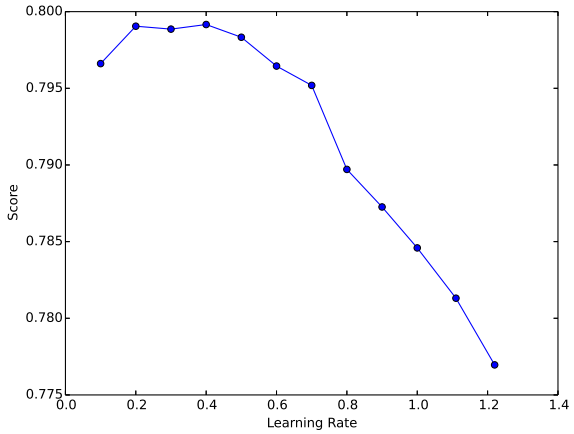

Figure 8: Plot showing gradient boosting learning rate against AUC score

with different kernel functions. One can see that they barely performed better than predicting all positives. Perhaps it was because the kernel functions used did not translate the input vectors into a vector space where the data is linearly separable. The team wasn't able to use linear or polynomial kernels as the training time was very high and the number of machines that needed to be trained was very large.

Kernel Function	AUC Score
Radial Basis Function	0.50797
Sigmoid	0.5

Table 3: SVM kernel functions and their scores

3.4 Deep Learning

The deep belief network predictions did not demonstrate great performance and the submission returned a classification accuracy of 58%. However, this was down due to the vast number of variable parameters present and the values that needed to be assigned to them. After some experimentation with 4 layers and a longer training period, a new DBN started to show promising results although the training time was becoming unrealistic.

3.5 Unsupervised outlier detection

The unsupervised outlier detection achieved an AUC score of 0.60081 which shows an improvement over predicting all positives but not a significant one. Certainly it does not perform as well as the tree ensemble methods.

4 Conclusion

This was a difficult problem that with noisy data that hampered progress slightly. It required careful thought to single out features that would allow uniquely identifying a driver. Apart from deciding on those, preprocessing and generating all of them takes a long time. The chosen features were used with various machine learning techniques, from deep neural networks to classic SVM to random forest classifiers, with varying success. The random forest with 500 estimators preformed the best achieving a score of 0.80542 on Kaggle. This is a decent score placing the team in the top third of the leaderboard, but compared to the highest result (0.97984) there is still much room for improvement. The techniques which were tried are well suited to the specific problem, but better features through preprocessing would likely be required to improve the score.

4.1 Further Work

One area which the team did not attempt to address is trip matching and road segment matching. Many drivers repeat the same trips or drive through the same roads. Identifying this would provide additional features which may hold some predictive power. Since the trips in the dataset are randomly rotated, trip matching is not a straight forward problem. One possible method would be to use the Ramer-Douglas-Peucker algorithm[8] for reducing the number of points on a curve an attempt to generate road segment features from these simplified trips.

Another area for improvement would be better choice of bin boundaries for the histograms. The bin boundaries were largely chosen based upon intuition from looking at various histograms of trips and drivers. A more scientific approach would have possibly yielded better results.

Additional work could include training a Convolutional Neural Network (CNN) on the raw (x, y) data instead of the hand-picked features. This type of deep learning model learns from grid-based data such as images and time series data.

Bibliography

[1] Jerome H Friedman. "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics* (2001), pp. 1189–1232.

[2] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[3] Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[4] Thomas Serre et al. "A quantitative theory of immediate visual recognition". In: *Progress in brain research* 165 (2007), pp. 33–56.

[5] Tai Sing Lee et al. "The role of the primary visual cortex in higher level vision". In: *Vision research* 38.15 (1998), pp. 2429–2454.

[6] Pascal Vincent et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion". In: *The Journal of Machine Learning Research* 11 (2010), pp. 3371–3408.

[7] Geoffrey Hinton. "Training products of experts by minimizing contrastive divergence". In: *Neural computation* 14.8 (2002), pp. 1771–1800.

[8] Urs Ramer. "An iterative procedure for the polygonal approximation of plane curves". In: *Computer Graphics and Image Processing* 1.3 (1972), pp. 244–256.