Vehicle Classification from Inductive Loop Signature.

Master of Science Dissertation

Surname: Bourgeois
First Name: Stéphane
Student Number: 12085745
Supervisor: Fabio Cuzzolin
Dissertation Module: CM 80
Course Title: MSc Computer Science
Date Submitted: 15 August 2014
Statement of originality

Except for those parts in which it is explicitly stated to the contrary, this project is my own work. It has not been submitted for any degree at this or any other academic or professional institution.

…………………………………………………………………………………………………………………………………………………………………………………………………………………..
Signature of Author  Date

Regulations Governing the Deposit and Use of Master of Science Dissertations in the Department of Computing and Communication Technologies, Oxford Brookes University.

1. The "top" copies of dissertations submitted in fulfilment of Master of Science course requirements shall normally be kept by the Department.

2. The author shall sign a declaration agreeing that, at the supervisor's discretion, the dissertation may be submitted in electronic form to any plagiarism checking service or tool.

3. The author shall sign a declaration agreeing that the dissertation be available for reading and copying in any form at the discretion of either the dissertation supervisor or in their absence the Head of Postgraduate Programmes, in accordance with 5 below.

4. The project supervisor shall safeguard the interests of the author by requiring persons who consult the dissertation to sign a declaration acknowledging the author's copyright.

5. Permission for anyone other than the author to reproduce in any form or photocopy any part of the dissertation must be obtained from the project supervisor, or in their absence the head of Postgraduate Programmes, who will give his/her permission for such reproduction only to the extent which he/she considers to be fair and reasonable.

I agree that this dissertation may be submitted in electronic form to any plagiarism checking service or tool at the discretion of my project supervisor in accordance with regulation 2 above.

I agree that this dissertation may be available for reading and photocopying at the discretion of my project supervisor or the Head of Postgraduate Programmes in accordance with regulation 5 above.

…………………………………………………………………………………………………………………………………………………………………………………………………………………..
Signature of Author  Date
This page intentionally left blank.
1 Abstract

Inductive loops are sensors that are widely deployed on road networks for the purpose of traffic data collection. Clearview Traffic Group has approached Oxford Brookes University to improve the performance of their current equipment.

Our aim is to classify vehicles in a 10 category scheme such as the SWISS10 from inductive loop signals. This level of classification is proving problematic with simple peak and valley detection algorithms. We believe that machine-learning algorithms are well suited for this task.

Vehicle-classification is a small player in the field of machine-learning, and literature published so-far only addresses a handful number of classes.

Equipment in the field runs on low-power embedded platforms, typically ARMv4T architecture, clocked at 8MHz, which delivers in the order of 10 MIPS.

We looked at two machine-learning algorithms: Support Vector Machines and Adaptive Boosting with decision stumps. We used the two most common algorithms for multi-class classification, One-versus-One and One-versus-Rest, and we looked at addressing class-imbalance with Under-sampling, Oversampling and SMOTE.

Support Vector Machines with the Gaussian kernel delivered the best results, with the smallest difference of performance between classes. An average f1-score per class of 90% was achieved with a maximum of 95% and a minimum of 85%.

The SMOTE algorithm proved useful to detect outliers and improve the results with the Polynomial Kernel.

The classifier needs to be implemented in fixed-point arithmetic for the embedded platform. We assessed that sufficient resolution can be obtained using 32 bits for variables, and 64 bits for multiply-accumulate operations.

Finally, an estimate of the ARM assembly instruction count was produced by derivation and concluded that our classifier computation time would be in the order of 120ms for 10 classes with an average 100 support vectors per class.
2 Acknowledgements

This project originates from a partnership between Clearview Traffic Group Limited and Oxford Brookes University. I would like to thank Chris Barnes from Clearview Traffic for initiating this project and Fabio Cuzzolin for presenting it in a format suitable for an MSc Dissertation.

The completion of this project would not have been possible without my family's moral and financial support, in particular, from my wife Sylvia.

I would like to thank my dissertation supervisor, Fabio Cuzzolin, for his guidance and advice through the entire project. In particular for his expert advice on the topic of machine learning and for setting clear boundaries to the scope of the project.

Finally, I need to thank the authors of the Numpy, Scipy and Scikit-learn libraries for the quality of their work and the clarity of their documentation.
**Illustration Index**

Illustration 1: Inductive Loop Road Layout..............................................................13
Illustration 2: Inductive Loop Signals.....................................................................14
Illustration 3: Support Vector Machine - Linear scenario.......................................16
Illustration 4: AdaBoost Algorithm..........................................................................18
Illustration 5: Frequency Analysis - Log amplitude / Log frequency plots...............25
Illustration 6: Signal difference between two loops..................................................27
Illustration 7: Class Frequency of Vehicles in the dataset.........................................28

Illustration 8: Stratified Shuffle Split........................................................................29
Illustration 9: Pre-processing for Adaptive Boosting................................................32
Illustration 10: Tail of preceding vehicle at start of signal.........................................38
Illustration 11: Head of following vehicle at tail of signal.........................................39
Illustration 12: SMOTE vectors in the Bus category..................................................39

Illustration 13: Pre-processing Flowchart.................................................................44
Illustration 14: Grid Search for C and gamma parameters.........................................45
This page intentionally left blank.
3 Introduction and Context information

3.1 Dissertation Rationale

Oxford Brookes University has been approached for consultation by Clearview Traffic, a manufacturer of traffic management and data solutions. Clearview design and manufacture traffic data collection systems. Inductive loop sensors are widely used for this application. However, classifying vehicles with high accuracy for certain European classification schemes, such as the SWISS 10, is proving to be a challenge.

The aim of the dissertation is to assess how machine-learning techniques can address this problem, assess the performance of a choice of algorithms, as well as their computational cost.

Our aim is to provide a classifier capable of running in real time on an embedded platform. Our assessment should also help evaluate if current systems, using an ARM7TDMI core at 8MHz, could benefit from a mid-life upgrade.

The subject originality is the fact that it is targeted for a low-power embedded application.

3.2 Vehicle Categories

The SWISS 10 classifies vehicles in 10 categories. One category, called “Special Artic.” will be unfamiliar for most readers, it is a lightweight articulated vehicle, of shorter length than the common EU 40 tonnes, 5 axles, articulated lorry.

We show video captures of each vehicle category.

1. Bus / Coach
2. Motorcycle
3.3 *Inductive Loop Detectors*

Inductive Loop Detectors are based on the change of inductance that occurs when a conductive object is placed in the proximity of a coil. The coil is driven by an oscillator, and the oscillator frequency changes as a conductive object is placed near the loop. Inductive loops are embedded under a groove made on the road surface.

![Illustration 1: Inductive Loop Road Layout](image)

Vehicles passing over the loops produce changes in the oscillators frequency, this change is sampled in a waveform called vehicle signature. The speed of the vehicle affects the duration of the signal. Dual loop installations allow to estimate the speed and length of the vehicle. The dimensions of the loop is around 2x2 meters, which has a low-pass effect on the signal.

Inductive Loop Detectors (ILDs) have been the most popular sensor since the 1960s. They are used both for the collection of traffic data and to drive automation of traffic lights. Although the current trend for new installations is to use Anisotropic Magnetoresistive (AMR) sensors, inductive loops are widely deployed on existing infrastructure.
These sensors are connected to cabinet on the side of the road that records and classifies the traffic. The main functional components are: (1) the LC oscillators of which the loops compose the inductive part, (2) D/A converters and associated circuit to record the frequency of the oscillators, (3) one or more processor boards to classify the signals and log the results, (4) telecoms interface / modem, (5) timekeeper to generate time stamps, (6) power supply and power management.

Our work is limited to part (3) of the system, in the algorithms performing the signal-
processing and classification. Frequency detection has already been performed in the data provided to us, we are provided with a demodulated signal with a DC component.

### 3.4 Machine Learning Classification

Classification is a branch of machine learning or pattern recognition, with foundations in signal processing and statistics. Machine learning is a well-established field, with success stories in machine vision, speech recognition, biomedicine, fraud detection. Vehicle classification is a small player in this field. The technology suited for our application is called supervised learning.

In supervised learning, the algorithm is presented with a set of signals that have already been classified by a human operator. The data comprises two parts: the signal samples, and the classification values. Supervised learning is performed in two stages: (1) the signals and the meta-data is fed in the algorithm, this is called the learning stage. (2) The algorithm is tested on new data, that was not part of the training set, to assess its performance. (3) Parameters are tuned to improve the results obtained in stage (2). Several iterations of training and tuning are performed, this is called cross-validation.

The aim is to derive a classifier that is not too complex in computation, is not too sensitive to noise and outliers, and provides good generalisation. Generalisation is an important aspect, and is the reason for cross-validation.

Following the literature review, we will hand-pick one or two machine-learning algorithms that appear to be good candidates for our problem.

For these algorithms we want to: (1) process the data to a format suitable for classification. Most classification algorithms expect the data to be normalised. This implies a preliminary analysis of the signal in the time and frequency domains. (2) tune the algorithm parameters and potential pre-processing options, (3) address issues of class-imbalance, noise and outliers.

Two algorithms are commonly used for classifying one-dimensional signals, these are Support Vector Machines and Adaptive Boosting. We will shortly describe them in the following chapters.
### 3.5 Introduction to Support Vector Machines

Support Vector Machines are a maximum margin classifier. They are used for binary classification, meaning data belonging to two classes only. The data is presented in the form of vectors of N dimensions and a classification label.

Support Vector Machines are not scale invariant and the data is normalised within the [0..1] or [-1..-1] interval. Each sample vector $x_i$ is associated with a classification label $y_i \in \{-1, +1\}$

The following diagram illustrates a scenario in 2 dimensions.

A Support Vector Machine will separate the data with a hyperplane illustrated by the green line, by finding the largest possible margin, illustrated by the dotted lines. The samples on the dotted line are called the support vectors. This is a constrained optimisation problem. The solution yields a classification function expressed as a linear function of the dot products with the support vectors.

For a vector $x$, we can express the decision function as
\[
d(x) = \sum_{i=1}^{N} \alpha_i y_i \langle x_i | x \rangle + b
\]

where \( x_i \) are the support vectors, \( y_i \) the classification labels, \( \alpha_i \) and \( b \) are the results of the optimisation.

Many problems are not linearly separable, in which case solutions can be found by projecting the data into a different space. As the solution is expressed as a function of the dot products only, it is not required to give an explicit function for the projection. Another function, called the Kernel function, is used instead of the dot product.

The two most common functions are the polynomial kernel and the Gaussian kernel. These are expressed for the polynomial kernel:

\[
K(x, x') = (\gamma \langle x | x' \rangle + c)^n
\]

for the Gaussian kernel:

\[
K(x, x') = e^{-\gamma \|x - x'\|^2}
\]

### 3.6 Introduction to Adaptive Boosting

Adaptive Boosting (AdaBoost) is a meta-algorithm. It is an iterative algorithm that combines the output of a large set of simple classifiers to produce a stronger classifier. The “strong” classifier is a linear combination, expressed as

\[
f(x) = \sum_{i=1}^{T} \alpha_i h_i(x)
\]

where each \( h_i(x) \) is a simple classifier. The decision function is simply the sign of \( f(x) \)

At the beginning of the iteration, each input vector is given an associated weight. Initially all weights are set to 1/N, where N is the number of dimensions. At each iteration, AdaBoost adjusts the weights to emphasize misclassified vectors from the previous iteration. AdaBoost then picks any simple classifier for which the weighted classification error is better than 0.5. The new component is assigned a vote (alpha) based on its associated error.

The iteration is stopped either when a set margin of error is reached, or at a maximum number of classifiers. At the end of the iteration, the values of the votes \( \alpha_i \) emphasize the most reliable classifiers.
The AdaBoost parameters are shown schematically in the following diagram.

As a large set of classifiers is used, individual classifiers need to be of low complexity. Commonly used classifiers are decision stumps. A decision stump only looks at a single component of the input vector \(x=(x_1, x_2, \ldots, x_n)\) to produce a classifier:

\[
h(x, \theta) = \text{sign}(ax_k - b)
\]

where the training parameters are \(\theta = \{k, a, b\}\)

Another commonly used classifier are random linear discriminants. The random parameters are chosen in advance, this technique is called a pool of random discriminants.

### 3.7 Embedded System implementation and Fixed Point Arithmetic

Our client expressed the requirement to have an estimate of the computation cost of the classifier. We are aiming at a system where the the training stage is performed on desktop hardware, in our case Intel x64 architecture. The classifier on the other hand is to run on an embedded platform. Our client current platform runs on an ARM7TDMI Processor Core, which supports the ARMv4 32bits and ARM Thumb 16 bits instruction sets. The platform is solar powered and is clocked at 8 MHz.
Even when considering more modern processor cores, the power requirements for a system in the field are such that we can't rely on floating point hardware. The classifier will need to be implemented in fixed-point arithmetic.

Our estimation involves: (1) derive an operations count from the classification experiments and parameters such as the number of classifiers, dimensions of input vectors, etc. (2) Estimate the required resolution for the fixed-point computation. This involves an estimation of the error propagation, and a prototype implementation of the classifier function in fixed-point arithmetic. (3) Estimate the instruction count and cpu cycle count for an implementation in ARM assembly language.

### 3.8 Aims and objectives

In the light of our client's brief, the aim of this dissertation is to:

1. Design classification algorithm(s) that can operate from the data available from existing infrastructure, ideally without the requirement to increase sampling rate.

2. Assess these solutions and compare their performance.

3. Estimate the computational complexity of the best solutions in order to assist our client in evaluating the implementation on embedded platforms.

In order to achieve this, we plan to:

1. Review current approaches in vehicle classification and one-dimensional signal classification (e.g. ECG, speech processing)

2. Choose a programming environment, signal processing and machine learning library that we believe to be suitable for prototyping.

3. Select candidate algorithms and implement those at a prototype level.

4. Choose a performance metric and cross-validation method relevant for the problem at hand.

5. Develop a software tool to tune and optimise the parameters of those classifiers.

6. Highlight the algorithms and parameters that offer the best performance.
7. Evaluate the computational complexity of the various solutions and draw a comparison of complexity versus accuracy of classification.

What is out of the scope of this dissertation:

1. User-friendly application with graphic user interface suitable for the non-programmer.

2. Implementation of classifier capable of running on embedded platform, e.g. using fixed-point arithmetic and written in C or ARM assembly.

3.9 Deliverables

1. Literature review

2. Signal analysis and implications for pre-processing.

3. Software tool to compare the performance of various machine learning algorithms and pre-processing methods.


5. Evaluation of the fixed-point implementation of the classifier. Evaluation of error propagation and required resolution.

6. Estimation of computational cost applicable to ARM architecture.
4 Available Resources and Literature

4.1 Machine learning textbooks and lecture notes

Our starting point was “Machine Learning in Action” (Harrington, 2012) which provides a hands-on approach in Python. Harrington covers the implementation of classic algorithms such as Platt's Sequential Minimal Optimization, but skips much of the mathematical derivations. A more rigorous approach can be found in “Pattern Recognition and Machine Learning” (Bishop, 2006).


4.2 Papers on Vehicle Classification

Vehicle classification is a small player in the field of machine learning. Only a handful of articles have been written between 2006 and 2010 on the subject. A majority of the literature is based on single loop inductive detectors and Artificial Neural Networks. The classification accuracy is around 90%, but for a smaller number of vehicle classes than in our case (e.g. motorcycle, car, small van, truck, bus). Although using a different sensor, articles using Anisotropic Magnetoresistive Sensors (AMR) are relevant to our application.

We summarise these results in the following table, including the type of detector (single, double inductive loop or AMR), pre-processing and choice of features for the input vector, size of the input vector (IV), classifier and training function, finally the number of classes and the average accuracy.

<table>
<thead>
<tr>
<th>Author</th>
<th>Det.</th>
<th>Pre-processing IV features</th>
<th>IV size</th>
<th>Classifier</th>
<th>Classes score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta &amp; Cinsdikici (2010)</td>
<td>2 ILD</td>
<td>DFT filtering, threshold, local maxima &amp; minima, principal component analysis.</td>
<td>17</td>
<td>3 layers Backpropagation Neural Network (BPNN) and RBFNN. SCG.</td>
<td>(5) 94%</td>
</tr>
<tr>
<td>Oliveira et al. (2010)</td>
<td>2 ILD</td>
<td>Signal maximum, energy, time of travel, number of local maxima.</td>
<td>8</td>
<td>3 layers Perceptron. Levenberg-Marquard.</td>
<td>(4) 92%</td>
</tr>
<tr>
<td>Author</td>
<td>Det.</td>
<td>Pre-processing IV features</td>
<td>IV size</td>
<td>Classifier</td>
<td>Classes score</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------</td>
<td>---------------------------------------------------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Kaewkamnerd et al. (2010)</td>
<td>AMR</td>
<td>Vehicle length, hill pattern peaks, energy distribution</td>
<td>5</td>
<td>Tree classifier + thresholds</td>
<td>(4) 81%</td>
</tr>
<tr>
<td>Feng &amp; Mingzhe (2009)</td>
<td>AMR</td>
<td>Length, mean, variance, peak, valley, energy.</td>
<td>6</td>
<td>Binary tree SVM (linear, polynomial, RBF kernel)</td>
<td>(5) 85%</td>
</tr>
<tr>
<td>Bajaj et al. (2007)</td>
<td>1 ILD</td>
<td>10 points waveform, histogram of 1st derivative</td>
<td>19</td>
<td>3 layers ANN. Genetic algorithm</td>
<td>(5) ?</td>
</tr>
<tr>
<td>Ki &amp; Baik (2006)</td>
<td>1 ILD</td>
<td>Normalized waveform.</td>
<td>19</td>
<td>3 layers BPNN</td>
<td>(5) 92%</td>
</tr>
</tbody>
</table>

No significant jump in classification accuracy seem to happen since the paper by Ki & Baik (2006). In particular, the attempt to extract statistical features of the signal and reduce the size of the input vector rarely improves the final accuracy.

We note an overwhelming use of Artificial Neural Networks and multi-layer perceptrons. This may be due to the fact that these publications have appeared before the release of the open source LIBSVM library that implements the Sequential Minimal Optimisation (SMO) algorithm for Support Vector Machines. (Chang & Lin, 2011)

One author, Kaewkamnerd et al. (2010) is targeting an embedded platform, built around a Texas Instruments MSP430 16-bits microcontroller. (Texas Instruments, 2014) Kaewkamnerd's team are using a decision tree classifier, based on low complexity features of the signal such as relative vehicle length, hill-pattern peak and distribution of energy. Their platform offers no hardware multiplication and overall less computing power than the architecture we are targeting.

Feng & Mingzhe (2009) are using Support Vector Machines arranged in a binary decision tree to classify four classes of vehicles from anisotropic magnetoresistive (AMR) sensor signals. The authors compare Linear, Polynomial and Gaussian kernels. Classification rates for large-truck, bus, van and car categories are respectively 86%, 80%, 81% and 89%.

Meta & Cinsdikici (2010) derive two types of parameters from inductive loop signals: (1) Principal component analysis (PCA) from which they retain 16 eigenvectors, (2) Statistical parameters derived from the waveform: median, mean, number of local maxima. A vector composed of the 16 PCA components and count of local maxima is fed into a three layer back-propagation neural network. The average recall rate for 5 vehicle classes is 94.2%.

Regardless of the classifier used, the main difference between authors is the complexity...
and type of pre-processing involved. Some authors prefer to extract the main features of
the signal at the pre-processing stage, with the aim to reduce the number of dimensions
of the input vectors.

4.3 Multi-class Algorithms and class imbalance

We found introductions to multi-class techniques and class imbalance both in journal
articles and in lecture notes.

Ryan Rifkin advocates that in most scenarios, the most commonly used multi-class
methods perform just as well as sophisticated algorithms. Namely, one-versus-all using
winner-takes-all strategy and the one-versus-one with max-wins voting. (Rifkin, 2008)

In another article, “In Defense of One-Vs-All Classification”, Rifkin argues that the
one-versus-all method is as effective as any. The Scikit-learn documentation on
“Multiclass and multilabel algorithms” also suggests that one-versus-all is a fair default
choice. (Rifkin & Klautau, 2004)

Class imbalance is a very rich topic. However, as pointed by Wang & Yao (2012), a
majority of articles only address two-class scenarios. I would add that a large number of
articles focus on outlier detection. This is relevant for applications in security and
biomedicine, where the focus is to detect rare diseases or intruders, but is less relevant
to us as we are concerned with performance over all vehicle classes.

Our starting point was a book chapter written by the author of the SMOTE algorithm,

Chawla states that “The main goal for learning from imbalanced datasets is to improve
the recall without hurting the precision.” He also notes that as precision and recall can
vary in conflicting directions, the f1-value is a good metric to assess the performance of
a classifier with unbalanced data.

We paid attention to class imbalance with Support Vector Machines. Although SVMs
are sensitive to data imbalance, they perform well for moderately skewed datasets (eg.
up to 1:100 ratio). Two methods stand out for their simplicity: (1) use of different error
costs per class, (2) oversampling and under-sampling of the data. (Batuwita & Paladey,
2013)
Our next step is to study the widely cited SMOTE (Synthetic Minority Over-Sampling Technique) algorithm. The algorithm is described with pseudo-code in the original paper by Chawla et al. (2002).

Akbani et al. (2004) expose the issues related to under-sampling and suggests to use SMOTE with different error costs on the positive and negative class.

### 4.4 Fixed point arithmetic and embedded systems

“Computer Organization and Design” (Patterson & Hennessy, 2013) provides an solid introduction to embedded systems programming. The author focuses on MIPS architecture which is close to our target platform. Appendix B of the 5th Edition provides ARMv4 instruction cycle timing information that we can use to estimate the cost of multiplications. (Sloss et al., 2013)

“The Neglected Art of Fixed-point Arithmetic” (Lauha, 2006) provides a good starting point on the subject, as well as the “Fixed Point Design” lecture notes. (Bolic, 2014)

Platform-specific information is available in an application note “Fixed Point Arithmetic on the ARM” (ARM Ltd, 1996)

For the Cortex M3 and M4 architectures, CPU cycle count of assembly operations are available on Technical reference manuals on the ARM website (ARM Ltd, 2014)
5 Methodology

5.1 Signal properties and pre-processing

Our first step is to look at the main properties of the signal such as the bandwidth, frequency response and signal-to-noise ratio. These characteristics tell us where the information is contained in the signal, and how to prevent losing this information during pre-processing.

5.1.1 Frequency analysis, Bandwidth

As described in the introduction, the relative dimensions of the inductive loops compared to the size of a vehicle has a low-pass effect on the signal. It is therefore useful to look at the signal in the frequency domain. The cutoff frequency is determined by the physical dimensions of the loop and the speed of the vehicle. We used Scipy Fourier Transforms to plot the spectral density of signals for each class of vehicle.

Illustration 5: Frequency Analysis - Log amplitude / Log frequency plots
These plots are logarithmic in amplitude and frequency. We can see that the signals decay in frequency, and are limited in bandwidth to around 30Hz. Cars travelling at a faster speed exhibit a bandwidth of around 40Hz. Our sampling frequency is 500Hz, the Nyquist frequency for our signal is around 60Hz. There is a potential for downsampling the signal at 8:1 ratio without losing information.

The following table shows sample vehicle durations, bandwidth and potential for downsampling.

<table>
<thead>
<tr>
<th>Category</th>
<th>Samples</th>
<th>Cutoff (Hz)</th>
<th>ratio</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Bus or Coach</td>
<td>250</td>
<td>30</td>
<td>8.33</td>
<td>30</td>
</tr>
<tr>
<td>3 Car or light van</td>
<td>100</td>
<td>60</td>
<td>4.17</td>
<td>24</td>
</tr>
<tr>
<td>4 Car + trailer</td>
<td>200</td>
<td>40</td>
<td>6.25</td>
<td>32</td>
</tr>
<tr>
<td>5 Heavy Van</td>
<td>150</td>
<td>30</td>
<td>8.33</td>
<td>18</td>
</tr>
<tr>
<td>6 Heavy Van + trailer</td>
<td>350</td>
<td>30</td>
<td>8.33</td>
<td>42</td>
</tr>
<tr>
<td>7 Special Artic</td>
<td>300</td>
<td>30</td>
<td>8.33</td>
<td>36</td>
</tr>
<tr>
<td>8 Rigid</td>
<td>300</td>
<td>30</td>
<td>8.33</td>
<td>36</td>
</tr>
<tr>
<td>9 Rigid + trailer</td>
<td>450</td>
<td>30</td>
<td>8.33</td>
<td>54</td>
</tr>
<tr>
<td>10 Artic</td>
<td>300</td>
<td>30</td>
<td>8.33</td>
<td>36</td>
</tr>
</tbody>
</table>

Mean 34.22

A waveform of 30 to 50 samples is a good candidate size for input vectors.

**5.1.2 Signal / Noise ratio and resolution**

Another property of the spectral density plots is the signal to noise ratio. The noise floor is at around -40dB for peak amplitude at -8dB, this is around 32dB resolution, which is
equivalent to **11 to 12 bits per sample.** Although our classification library uses double precision floating point, this value is relevant for a fixed point implementation.

### 5.1.3 Signal difference between the two loops

The two inductive loop signals are produced from two identical coils driven by two oscillators at two different frequencies. A question is whether there is a difference between the two signals other than in amplitude, or whether the two signals are copies of each other, and the only information is the time-delay between the signals.

The time-delay is of major significance as it enables us to estimate the vehicle speed and vehicle length.

Looking at signal plots normalised in amplitude, it would appear that the two channels are identical. The above example exhibits the largest amount of variation observed on a set of signals. The difference may be attributed to the vehicle acceleration. Other visible differences are attributable to noise in the signal.

### 5.2 Properties of the dataset and Cross-validation

We have a small data set by machine learning standards. The number of samples per vehicle class is also unbalanced. The classes and sample count are listed in the table below.
We decided to ignore the motorbike category, and created alternative sets of XML metadata files to correct vehicles that had been misclassified (for which Clearview issued a correction note).

The number of vehicle manufacturers is limited, and the practice of platform sharing amongst manufacturers makes it likely that near-identical signals are repeated in the data. With repeated signals, we run the danger of overfitting the classifier. Repeated signals could also introduce a bias in the cross-validation, in particular with Leave-One-Out method. The low number of class 1 (bus or coach) samples means that if we choose Stratified k-fold, we are constrained to use k≤5. We chose the Stratified Shuffle Split algorithm. This allows us to control the ratio of samples in the training vs.

<table>
<thead>
<tr>
<th>class</th>
<th>vehicle type</th>
<th>samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bus or Coach</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Motorbike</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Car or light van</td>
<td>1102</td>
</tr>
<tr>
<td>4</td>
<td>Car or light van plus trailer</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>Heavy van or minibus</td>
<td>158</td>
</tr>
<tr>
<td>6</td>
<td>Heavy van or minibus plus trailer</td>
<td>56</td>
</tr>
<tr>
<td>7</td>
<td>Special artic</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>Rigid</td>
<td>144</td>
</tr>
<tr>
<td>9</td>
<td>Rigid plus trailer</td>
<td>127</td>
</tr>
<tr>
<td>10</td>
<td>Articulated Lorry</td>
<td>318</td>
</tr>
</tbody>
</table>

*Illustration 7: Class Frequency of Vehicles in the dataset.*
the testing set independently of the number of iterations and class imbalance.

A drawback of the Shuffle Split algorithm is Monte-Carlo variation in case of a low number of iterations (20 to 100). This is apparent when performing a grid search to optimise Support Vector Machine parameters C and $\gamma$.

We ran 10 repeated Shuffle Split estimations, on RBF and polynomial kernel SVMs, with 20, 100 and 1000 iterations. This enabled us to evaluate the variance and standard deviation affecting our results.

1000 Shuffle Split iterations produced a standard deviation of 0.1% for the worst performing class. We will therefore run our experiments with a Stratified Shuffle Split of 1000 iterations, an 80% training set and 20% test set.

### 5.3 Performance metric

In a multi-class context, classifier performance is usually reported in a confusion matrix and a report of the precision, recall and f1-score per class. These reports yield too much data to rank the performance of algorithms. An example of such classification report is illustrated below:

```python
>>> svm.fastclf('vecC500_64w1-n', n_iter=100, multi_class='ovr', C=500, gamma=0.5, kernel='poly', degree=3, verbose=1)

<table>
<thead>
<tr>
<th>class</th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus</td>
<td>0.67</td>
<td>0.75</td>
<td>0.71</td>
<td>100</td>
</tr>
<tr>
<td>Car</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>10000</td>
</tr>
<tr>
<td>Car +T</td>
<td>0.85</td>
<td>0.87</td>
<td>0.86</td>
<td>800</td>
</tr>
<tr>
<td>Heavy van</td>
<td>0.82</td>
<td>0.86</td>
<td>0.84</td>
<td>3200</td>
</tr>
<tr>
<td>Hvy van +T</td>
<td>0.85</td>
<td>0.80</td>
<td>0.82</td>
<td>1100</td>
</tr>
<tr>
<td>Spec. Artic</td>
<td>0.75</td>
<td>0.70</td>
<td>0.72</td>
<td>300</td>
</tr>
<tr>
<td>Rigid</td>
<td>0.92</td>
<td>0.90</td>
<td>0.91</td>
<td>2900</td>
</tr>
<tr>
<td>Rigid +T</td>
<td>0.95</td>
<td>0.90</td>
<td>0.93</td>
<td>2500</td>
</tr>
<tr>
<td>Artic</td>
<td>0.94</td>
<td>0.96</td>
<td>0.95</td>
<td>6400</td>
</tr>
<tr>
<td>avg / total</td>
<td>0.93</td>
<td>0.93</td>
<td>0.93</td>
<td>27300</td>
</tr>
</tbody>
</table>
```

Illustration 8: Stratified Shuffle Split

Vehicle Classification from Inductive Loop Signature
From the literature on class-imbalance, we know that the f1-score is the best metric to reflect overall performance. The f1-score is the harmonic average of the precision and recall, which produces a “pessimistic” average value.

The default implementation reports average scores weighted by the number of samples. This introduces a bias as cars are over-represented in our data, and are particularly easy to classify.

We chose two parameters to rank the performance of the classifiers: 1. the unweighted average of the f1-score per class, 2. the minimum f1-score. The first value gives an overall representation of the performance across all classes, the second gives us an indication of the worst performing class.

When comparing class-imbalance methods, we will use the f1-score per class.

5.4 Choice of Classification algorithms

Our choice of algorithms is based on the literature review, and advice from our supervisor, they are Support Vector Machines and Adaptive Boosting.

We have already described these algorithms briefly in the introduction.

Support Vector Machines offer several advantages: (1) they scale better than k Nearest-neighbours, (2) come with theoretical guarantees of performance, (3) have shown empirically good performance with applications in many fields (biomedicine, image recognition, speech processing) (Weston, 2004)

Provided we use simple enough base classifiers, Adaptive Boosting is of interest as we aim for low computational requirements of the classifier function. The characteristics of Adaptive Boosting are: (1) few parameters to adjust, (2) less sensible to over-fitting than other methods, (3) sensitive to noise and outliers. (Harrington, 2012)

We will test both Support Vector Machines and Adaptive Boosting.


5.5 Classification with Support Vector Machines

With Support Vector Machines, the following parameters are available to us:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temporal window</td>
<td>(1) single channel start/end, (2) channel 1 start / channel 2 end, (3) normalized to vehicle length.</td>
</tr>
<tr>
<td>Number of samples in the input vector</td>
<td>128, 64, 32, 16.</td>
</tr>
<tr>
<td>Number of channels</td>
<td>1 or 2. Signal analysis points to using 1 channel only.</td>
</tr>
<tr>
<td>Kernel Function</td>
<td>Gaussian (RBF), Polynomial (degree 2 or 3), Linear</td>
</tr>
<tr>
<td>Multi-class method</td>
<td>One-versus-one, One-versus-all</td>
</tr>
</tbody>
</table>

For each of these scenarios, we wish to train the penalty parameter C and kernel parameter γ. We also wish to reduce the computational complexity of the classifier by the choice of vector size and kernel function, and assess the impact on the performance.

As a starting point, I followed recommendations from my thesis director and from “A Practical Guide to Support Vector Classification” (Hsu et al., 2003)

1. Conduct simple scaling of the data.
2. Consider various signal windows.
3. Consider the RBF kernel first.
4. Use cross validation to search for the optimum C and γ
5. Compare One-versus-One and One-versus-All.

5.6 Classification with Adaptive Boosting

Adaptive Boosting is a meta-algorithm that improves the performance of a set of base classifiers. We are going to use decision stumps, that is the simplest type of classifier.

Two types of input vectors can be used for boosting: (1) the normalized signals as prepared for the Kernel SVMs, (2) after normalising the signal, use a pool of linear discriminants. In our case, we use difference in amplitude between 2 samples taken from a list of random temporal values.
The parameters to adjust are shown in green on the above diagram, and summarised as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temporal window</td>
<td>(1) single channel start/end, (2) channel 1 start / channel 2 end, (3) normalized to vehicle length.</td>
</tr>
<tr>
<td>Type of input vector</td>
<td>Waveform + statistics / Linear discriminants</td>
</tr>
<tr>
<td>Number of samples in the signal</td>
<td>32, 64</td>
</tr>
<tr>
<td>Number of random discriminants</td>
<td>128, 256, 512</td>
</tr>
<tr>
<td>Learning rate</td>
<td>Adjust starting at 1.0</td>
</tr>
<tr>
<td>Number of base classifiers</td>
<td>64, 128, 256</td>
</tr>
<tr>
<td>Multi-class method</td>
<td>One-versus-one, One-versus-all</td>
</tr>
</tbody>
</table>

5.7 Multi-class method and class imbalance

Most classification algorithms are intended for binary classification. There are well-known methods to extend classification to multiple classes. The most commonly used are the one-versus-all method using winner-takes-all strategy, and the one-versus-one method with max-wins voting.

There is a cottage industry of sophisticated multi-class algorithms, as well as a number of authors to claim that they perform just as well as the common methods cited above. (Rifkin, 2008; Rifkin & Klautau, 2004)
We will use both one-versus-one and one-versus-all methods and compare the results.

We reviewed three methods in the literature to address class imbalance problems: (1) random under-sampling, (2) over-sampling and (3) Synthetic Minority Oversampling Technique (SMOTE). We will test and compare all three, we will also combine random under-sampling with any over-sampling methods.
6 Software Implementation

6.1 Specification

We drew the following specification from our client's requirements.

**Required:**

1. Read the data in the format presented by Clearview (XML + csv)
2. Perform signal processing in modular steps.
3. Assemble feature vectors, select classification algorithm and parameters in modular steps.
4. Create data sets for cross-validation, and collect classification scores in a database.
5. User can control parameters of steps 2 to 4 from configuration files (e.g. XML)
6. Automation to run tests without user input and collect score in log files / database
7. Log all information useful to evaluate computation complexity (e.g. number of support vectors, vector dimensions, number of classifiers, etc.)
8. Serialize data between steps 1 and 3 to accelerate (and distribute) computation.

**Desirable:**

1. Plotting / visualisation of waveforms, results, scores...
2. Robust pre-processing. Simple method to blacklist outliers.
3. Output logs to detect errors and compare our results with CCTV imagery.

**Outside the scope of this specification:**

1. Embedded system classifier prototype / implementation.
2. User-friendly GUI suitable for a non-programmer.
6.2 Programming language and environment

6.2.1 Choice of programming language and libraries
We don't know, at the onset of this project, which algorithm or combination of algorithms will deliver the best outcome. We have therefore opted for an incremental prototyping methodology.

Candidate languages offering machine learning libraries are Matlab / GNU Octave, Python + Numpy + SciPy + Scikit-learn and Java + Weka. We chose to use Python + Scikit-learn.

This choice was motivated by two factors. First, Python stood out as an environment that offers a comprehensive set of solutions out-of-the-box: XML and csv parsing, object serialization, database interaction and unit testing are built in the language. NumPy and Scipy offer scientific computation and signal-processing, Scikit-learn is a relatively comprehensive machine-learning library. Second, preliminary tests and prototyping done at the stage of writing the Dissertation Proposal proved very encouraging.

6.2.2 Configuration and development environment
For compatibility purposes with the Scipy, Numpy, Scikit-learn libraries we use Python version 2.7.3 instead of the new Python 3.0 release.

Having more development experience in the Linux environment, we chose to work under Debian stable 64 bits (Wheezy). The development is done for cross-platform compatibility, we use the Python os.path module for pathname manipulation, and validate on a Windows 7 32 bits platform.

The Scikit-learn library in the Debian repository is out-of-date, we are using the releases from the NeuroDebian repository (Hanke & Halchenko, 2014)

For the Windows 7 platform, we used the binaries distributed by Christoph Gohlke (2014), Laboratory for Fluorescence Dynamics, University of California, Irvine.
6.3 Functions Implemented

Our implementation offers the following functions:

1. Serialization of the raw signal waveforms supplied to us in .csv format
2. Pre-processing of input vectors for Kernel SVM or AdaBoost. Serialization of the input vectors to speed-up classification.
3. Interactive classification from command line functions. This allows to run short tests with a single set of parameters.
4. Grid search for preliminary tuning of Kernel SVM parameters C and gamma
5. Batch mode classification. This runs multiple classifications in sequence according to parameters set in a JSON descriptor file. The user can select parameters affecting pre-processing, classifier, cross-validation, multi-class method, oversampling and under-sampling.
6. The batch mode classifier outputs a csv file to be imported in a spreadsheet. This file displays all the parameters, and produces f1-scores per class, average and minimum f1.
7. Auxiliary functions were written for plotting waveforms and testing the behaviour of the SMOTE algorithm.
8. Unit tests ensure correct parsing of the XML metadata supplied by Clearview.

6.4 Testing and verification

We summarise our test coverage in the following table.

<table>
<thead>
<tr>
<th>Function</th>
<th>Test type</th>
<th>Test strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML parsing</td>
<td>Unit Test</td>
<td>XML data created for testing. Unit test exercises parser with test data.</td>
</tr>
<tr>
<td>Parsing of csv data</td>
<td>Manual</td>
<td>Waveform plots. Visual comparison with ClearView desktop app. (PerformanceLoopDataCapture)</td>
</tr>
<tr>
<td>Pre-processing</td>
<td>Manual</td>
<td>Waveform and input-vector plots.</td>
</tr>
<tr>
<td>Frequency analysis</td>
<td>Manual</td>
<td>Option to inject pilot tones in the signal.</td>
</tr>
<tr>
<td>Classification</td>
<td>Incremental</td>
<td>Options to limit the number of samples and number of classes. Classification was tested with increasingly larger datasets, on two classes, then multi-class.</td>
</tr>
</tbody>
</table>
### Function Test type Test strategy
---
Cross-validation Manual Results are consistent when using Stratified k-fold and Stratified Shuffle Split with comparable test set ratio (e.g. k = 5, test set = 20%)
SMOTE Manual Test vectors included in module + verbose flag enable to assess: (1) correct choice of k-Nearest Neighbours, (2) linear interpolation between test vectors.
Batch processing Black-box Test harness to exercise: (1) change in each input parameter, (2) correct parsing of the JSON input file, (3) correct call of pre-processing function. The option to seed the pseudo-random number generators ensures the test results are repeatable.

We wrote unit tests to verify the parsing of XML metadata.

Parsing of the .csv data and signal-processing was tested by plotting waveform and spectrum input/outputs. Those waveforms could be compared with video imagery and waveforms displayed on ClearView desktop application. Tests were run on sample data and on test signals (e.g. sine waves, ramps).

Division-by-zero errors were reported during pre-processing. This was caused by conditions overlooked in the specification: some signals supplied by Clearview contained portions of the preceding or following vehicle. Some signals of long vehicles could also be clipped. A test condition to report errors in vehicle length or speed was developed, as well as a more robust algorithm to detect the start and stop of the waveform.

![Illustration 10: Tail of preceding vehicle at start of signal](1333.0-20130823_091421.csv --- 9 : Rigid +T)
The SMOTE implementation was tested in 2 ways: test vectors were used to assess the correct choice of the k Nearest Neighbours and interpolation of vector values. Waveform plots enables to assess the behaviour of the SMOTE algorithm on our data.

Finally, the batch-processor was tested using a black-box test harness. The option to seed any pseudo-random number generator used in Cross-validation, AdaBoost and
SMOTE, made the output results repeatable. The test harness verifies the repeatability of results with identical inputs and exercises each input parameter, one at a time.

6.5 Usage Example and Documentation.

Finally we want to illustrate the use of our application with a short example. The choice and meaning of the parameters is explained in the next section. We are here to demonstrate how the application is used in practice.

In this scenario, we want to tune the gamma parameter of a Support vector machine classifier. We wish to use a signal window of type “I”, an input vector of 32 samples. For the classifier, we wish to use the RBF kernel with a soft margin C=100 and gamma values around 0.25.

The first step is to serialize csv signal files into a more practical format. This is done only once for the entire dataset, and is independent of the class attributes in the meta-data. This means the user can build multiple sub-sets of meta-data, e.g. to change the class of the vehicles, without having to repeat this step. Note the presets for the list of XML meta-data files and the list of vehicle classes.

```
$ python
>>> import preproc as pre
>>> pre.XMLA
['20130823_091421.xml', '20130823_094746.xml', '20130823_102154.xml']
>>> pre.CLASSES
[1, 3, 4, 5, 6, 7, 8, 9, 10]
>>> pre.csv_serialize(pre.XMLA, 'clearA', class_list=pre.CLASSES, limit=0)
WARNING: missing class or data tag in 20130823_094746.xml at time PT9M27.6S
WARNING: missing class or data tag in 20130823_102154.xml at time PT50M40.7S
```

This step writes a file `clearA.pkl` that we can refer to for the rest of the operations and issues warnings for malformed XML tags.

We enter the parameters for both pre-processing and classification in a file containing a JSON object. Global parameters such as file paths are also specified in this file. Note that the processor behaves as a state machine. Unchanged arguments are not repeated from one run to the next.
We can then run the batch processor from the console

```
$ ./batch_svm.py test/batch_demo.json
Calculating input vectors...
loading waveform data from ./pkldata/clearA.pkl
WARNING: missing class or data tag in 20130823_094746.xml at time PT9M27.6S
WARNING: missing class or data tag in 20130823_102154_C.xml at time PT50M40.7S
calculating feature vectors...
1 / 3 --- time: 136.6
2 / 3 --- time: 137.8
3 / 3 --- time: 145.6
```

This creates a csv file to be imported in a Spreadsheet application. All the parameters are reported in the csv. This avoids any confusion regarding parameters omitted in the JSON object, that were set to their default values.

For further documentation, the code contains inline documentation using the Python

Vehicle Classification from Inductive Loop Signature
Docstring Conventions and the format specified in the Google Python Style Guide. This provides a consistent framework for documenting function arguments and types, and returned values.

The black-box tests (test_batch_boost.py and test_batch_svm.py) also demonstrate which functions are called by the batch processors, and the correspondence between console use and the JSON object.
7 Experimental Results

7.1 Pre-processing

Support Vector Machines are not scale invariant. We therefore need to scale the data both in amplitude and in the time-domain.

Our pre-processing algorithm is as follows:

1. The data supplied to us was deliberately sampled at 2kHz. In the light of signal analysis, it was decimated to 500Hz, which is the sampling frequency of equipment in the field.

2. Remove DC component. 64 samples were retained at the head of channel 2 and tail of channel 1 and averaged. The minimum of the the two averages were retained.

3. Detect the max. value for each channel and normalise the signal to max. amplitude = 1

4. Detect signal start / end by searching for the first and last sample with an amplitude threshold of 10% of the maximum amplitude.

5. If the result returns the first or last sample, we assume that the affected channel doesn't start or end on the DC average. We evaluate the time delay between the channels, and scan for a new start or end. The exact algorithm is detailed below.

6. From the start and end of both channels, we compute the average time-delay between channels and a estimate of the vehicle length.

7. We discard channel 1 which has the lowest amplitude and worst S/N ratio.

8. Resample the signal to a fixed-length window using nearest-neighbour lookup. This would normally lead to aliasing, but we know that our signal is bandwidth limited. At a later stage we will review this choice and introduce a low-pass filter to reduce the wideband noise and prevent aliasing.

9. Assemble a feature vector with: max. amplitude, vehicle length estimate, resampled signal. All the components in this vector are normalised to the [0 .. 1]
We made 3 choices of signal windows, with a different timing of the start and stop of the signal.

<table>
<thead>
<tr>
<th>Window type</th>
<th>starts</th>
<th>ends</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Start of channel 2</td>
<td>End of channel 2</td>
</tr>
<tr>
<td>2</td>
<td>Start of channel 1 (before chan 2)</td>
<td>End of channel 2</td>
</tr>
<tr>
<td>3</td>
<td>Start of channel 2</td>
<td>Proportional to vehicle length</td>
</tr>
</tbody>
</table>

The complete pre-processing is summarised in the following diagram.

*Illustration 13: Pre-processing Flowchart*
7.2 First approach to Support Vector Machine Classification

Cost factor C and gamma parameter have been tuned using grid search for all the following tests. Rough values are obtained first using a grid search for a logarithmic range of C from 0.1 to 50000 and gamma from 0.0001 to 5.0.

A more narrow search is then produced by the batch processing script. This also returns f1-score per class, minimum and average f1-score.

Our choice of classifier is determined as follows: (1) shortlist the C and gamma producing maxima in the average f1-score. (2) within comparable average f1-score, pick the best minimum f1-score.

For this first round of tests, we made two significant changes to the data: (1) the number of cars was limited to 500. This reduces class imbalance by a crude form of random under-sampling. (2) The bus class is represented by a handful of samples, one sample representing a bus with trailer was excluded from the dataset. These changes will be reverted when we address class imbalance and outliers.

7.2.1 Comparison of window type and vector size (with RBF Kernel)
The best results are obtained with a window type “1”, which selects the start and end of a single channel. The optimum vector size is 32 samples. This is consistent with our study of the signal bandwidth. 32 samples are enough to capture the signal information and lose redundant data.

### 7.2.2 Comparison of kernel functions


<table>
<thead>
<tr>
<th>vec. size</th>
<th>f1 average</th>
<th>f1 minimum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rbf</td>
<td>poly d=3</td>
</tr>
<tr>
<td>128</td>
<td>0.880</td>
<td>0.877</td>
</tr>
<tr>
<td>64</td>
<td>0.895</td>
<td>0.883</td>
</tr>
<tr>
<td>32</td>
<td><strong>0.899</strong></td>
<td><strong>0.895</strong></td>
</tr>
<tr>
<td>16</td>
<td>0.897</td>
<td>0.886</td>
</tr>
</tbody>
</table>
The Radial Basis Function kernel gives the best performance, this is consistent with general advice from textbooks. A degree 3 polynomial kernel scores second best. This can be of interest for an embedded implementation.

### 7.2.3 One-versus-one and One-versus-all

Support Vector Classifier. Kernel: RBF or 3rd degree polynomial, tuned C and gamma. Stratified Shuffle Split, training = 80%, testing = 20%, 1000 iterations. Multi-class method: one-versus-one or one-versus-all.

<table>
<thead>
<tr>
<th>vec size</th>
<th>RBF</th>
<th>RBF ovr</th>
<th>poly</th>
<th>poly ovr</th>
<th>RBF</th>
<th>RBF ovr</th>
<th>poly d=3</th>
<th>poly ovr</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>0.880</td>
<td>0.885</td>
<td>0.877</td>
<td>0.874</td>
<td>0.755</td>
<td>0.805</td>
<td>0.733</td>
<td>0.744</td>
</tr>
<tr>
<td>64</td>
<td>0.895</td>
<td>0.893</td>
<td>0.883</td>
<td>0.881</td>
<td>0.770</td>
<td>0.818</td>
<td>0.750</td>
<td>0.724</td>
</tr>
<tr>
<td>32</td>
<td><strong>0.899</strong></td>
<td><strong>0.902</strong></td>
<td>0.895</td>
<td>0.872</td>
<td><strong>0.804</strong></td>
<td><strong>0.807</strong></td>
<td>0.764</td>
<td>0.713</td>
</tr>
<tr>
<td>16</td>
<td>0.897</td>
<td>0.894</td>
<td>0.886</td>
<td>0.865</td>
<td>0.742</td>
<td>0.743</td>
<td>0.739</td>
<td>0.741</td>
</tr>
</tbody>
</table>
The One-vs-Rest method compares favourably with One-versus-one as long as we use the RBF kernel. However, the performance drops with a polynomial kernel. A possible interpretation is that the shape of the decision boundary becomes more complex in the One-vs-Rest scenario, and the RBF kernel has more capacity to adapt to this new boundary.

The decisive factor for an embedded implementation is the total number of support vectors involved in the multi-class computation. The number of support vectors is smaller for each One-versus-one individual classifier, but the number of computations for n classes is \( n \times (n-1)/2 \)

For a vector size = 32, One-vs-One, the number of support vectors per class is [4, 51, 28, 44, 33, 11, 53, 61, 102], for One-vs-Rest [29, 48, 72, 94, 91, 55, 93, 127, 159]

A comparison of computational cost is exposed in section 6 of the report.

### 7.2.4 Best Performers

The performance of the three best classifiers is summarised in the following table.

<table>
<thead>
<tr>
<th>window</th>
<th>vec. size</th>
<th>kernel</th>
<th>C</th>
<th>gamma</th>
<th>f1-average</th>
<th>f1-minimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32</td>
<td>RBF</td>
<td>500</td>
<td>1.4</td>
<td>OvR</td>
<td>0.902</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
<td>RBF</td>
<td>500</td>
<td>1.4</td>
<td>OvO</td>
<td>0.899</td>
</tr>
<tr>
<td>1</td>
<td>32 Poly d=3</td>
<td>500</td>
<td>0.1</td>
<td>OvO</td>
<td>0.895</td>
<td></td>
</tr>
</tbody>
</table>

Vehicle Classification from Inductive Loop Signature
The f1-score for each class of vehicle is summarised in the following table and graph.

<table>
<thead>
<tr>
<th></th>
<th>Bus</th>
<th>Car</th>
<th>Car+T</th>
<th>Van</th>
<th>Van+T</th>
<th>Sp.Artic</th>
<th>Rigid</th>
<th>Rig+T</th>
<th>Artic</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBF OvR</td>
<td>0.919</td>
<td>0.969</td>
<td>0.884</td>
<td>0.879</td>
<td>0.848</td>
<td>0.807</td>
<td>0.926</td>
<td>0.937</td>
<td>0.956</td>
</tr>
<tr>
<td>RBF OvO</td>
<td>0.868</td>
<td>0.971</td>
<td>0.884</td>
<td>0.895</td>
<td>0.839</td>
<td>0.795</td>
<td>0.933</td>
<td>0.940</td>
<td>0.957</td>
</tr>
<tr>
<td>Poly 3 OvC</td>
<td>0.858</td>
<td>0.970</td>
<td>0.885</td>
<td>0.898</td>
<td>0.854</td>
<td>0.764</td>
<td>0.948</td>
<td>0.928</td>
<td>0.955</td>
</tr>
</tbody>
</table>

The best classifiers were all obtained with a vector of 32 samples and a type “1” window (single channel 10% amplitude threshold start/stop). The RBF kernel performs best for the least represented classes. The polynomial degree 3 kernel gives acceptable results and should be kept as a contestant in the interest of lower computational cost.
7.3 Adaptive Boosting

7.3.1 Input Vectors and parameters for AdaBoost

For Adaptive Boosting we assembled two types of input vectors based on the pre-processing done for support vector machines. One approach is derive statistics from the signal, this is inspired by the approach of Meta & Cinsdikici (2010)

1. Using the previous input vectors with the 3 window types and signal sizes of 128 and 64 samples. We added signal statistics derived as follows:
   
   1.1. Use a 128 samples window of type “1”.
   1.2. Divide the signal in 3 equal length time segments (approx. 43 samples)
   1.3. For each segment, compute min, max, mean, variance, skewness, kurtosis

The input vector is composed of these statistics, appended to the normalised signal data.

2. Build a pool of random discriminants from the signal. These are composed of 128 or 512 differences between two samples taken at random time positions in the signal. Using the notation presented at the introduction, the decision function for this classifier can be expressed as follows.

   For an input vector \( x = (x_1, x_2, ..., x_n) \) we derive a classifier from two components

   \[
   h(x, \theta) = \text{sign}(u(x_j - x_k) - b)
   \]

   where the training parameters are \( \theta = \{j, k, u, b\} \) and \( u \in \{-1, +1\} \)

3. In both cases, the vehicle length estimate and maximum amplitude are added to the input vector

The input vector signal flow is illustrated below:
We are using AdaBoost with a decision stump classifier, three parameters can be tuned: the maximum number of estimators, the learning rate, and the type of algorithm SAMME or SAMME.R.

The SAMME and SAMME.R are multi-class algorithms. However we couldn't get decent results with our data, as our base classifiers are not intended for multi-class output. The best results were obtained with the One-versus-one method, and SAMME or SAMME.R used on two class decision.

### 7.3.2 Results

The SAMME.R algorithm converges faster with our data.

The training time for AdaBoost is an order of magnitude higher than for Support Vector Machines.

Results stop improving for a number of estimators larger than 256. Unlike support vector machines, windows of type 2 and 3 perform better than window 1. These windows feature sequences of zeros proportional to the vehicle length.

We summarise our best performers in the following table. Results are comparable to Support Vector Machines with a Linear Kernel.

<table>
<thead>
<tr>
<th>Input Vector</th>
<th>Window</th>
<th>samples</th>
<th>IV length</th>
<th>Multi cl</th>
<th># estim</th>
<th>L. rate</th>
<th>f1-avg</th>
<th>f1-min</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal + statistics</td>
<td>3</td>
<td>64</td>
<td>82</td>
<td>OvO</td>
<td>32</td>
<td>0.71</td>
<td>0.869</td>
<td>0.660</td>
</tr>
<tr>
<td>Signal + statistics</td>
<td>3</td>
<td>64</td>
<td>82</td>
<td>OvO</td>
<td>128</td>
<td>0.5</td>
<td>0.869</td>
<td>0.636</td>
</tr>
<tr>
<td>Linear discriminants</td>
<td>2</td>
<td>64</td>
<td>256</td>
<td>OvO</td>
<td>256</td>
<td>1</td>
<td>0.853</td>
<td>0.656</td>
</tr>
</tbody>
</table>
The f1-score per class shows more difficulties with the Bus/Coach (class 1) and Special Artic. (class 7) categories. We include results from Support Vector Machines for comparison.

<table>
<thead>
<tr>
<th></th>
<th>Bus</th>
<th>Car</th>
<th>Car+T</th>
<th>Van</th>
<th>Van+T</th>
<th>Sp.Artic</th>
<th>Rigid</th>
<th>Rig+T</th>
<th>Artic</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM RBF OvR</td>
<td>0.930</td>
<td>0.969</td>
<td>0.883</td>
<td>0.878</td>
<td>0.844</td>
<td>0.813</td>
<td>0.926</td>
<td>0.936</td>
<td>0.956</td>
</tr>
<tr>
<td>SVM Poly OvO</td>
<td>0.865</td>
<td>0.971</td>
<td>0.888</td>
<td>0.895</td>
<td>0.853</td>
<td>0.763</td>
<td>0.938</td>
<td>0.932</td>
<td>0.953</td>
</tr>
<tr>
<td>SVM Poly OvR</td>
<td>0.768</td>
<td>0.973</td>
<td>0.894</td>
<td>0.890</td>
<td>0.843</td>
<td>0.732</td>
<td>0.933</td>
<td>0.922</td>
<td>0.954</td>
</tr>
<tr>
<td>AdaBoost stats</td>
<td>0.713</td>
<td>0.972</td>
<td>0.898</td>
<td>0.890</td>
<td>0.854</td>
<td>0.660</td>
<td>0.932</td>
<td>0.936</td>
<td>0.963</td>
</tr>
<tr>
<td>AdaBoost lin disc</td>
<td>0.738</td>
<td>0.970</td>
<td>0.828</td>
<td>0.874</td>
<td>0.803</td>
<td>0.656</td>
<td>0.921</td>
<td>0.932</td>
<td>0.951</td>
</tr>
</tbody>
</table>

At this stage we decide to leave Adaptive Boosting aside and focus on Support Vector Machines.
7.4 Class Imbalance

In the previous tests, there appears to be a correlation between classification performance and the number of samples per class. On the whole, less-represented classes are harder to classify.

Two measures were already taken in the previous tests to reduce the effect of class imbalance: (1) the number of cars was limited to first 500 samples, which is a form of random under-sampling. (2) class weights were set to a value inversely proportional to class frequency. This is the recommended default choice for unbalanced data.

We are going to review these choices and look at other methods to address class imbalance with three classifiers: the RBF kernel with One-versus-all, the polynomial kernel with One-versus-all and One-versus-one.

This choice is motivated by the imperative of low computational cost. The RBF kernel support vector machine performed equally well with OvO and OvR method, we are therefore focusing on OvR. The polynomial kernel didn't perform well in OvR, we will assess how much this could be due to class imbalance.

In the following tests, our metric will be the f1-score per class.

7.4.1 Random Under-sampling and Class Weights

Under-sampling should be used with caution, as it introduces the risk of information loss.

In our case, cars are the most represented class of vehicles, and waveforms of cars (and motorcycles) present a very high regularity in the signal. Waveforms of other classes show less regularity, as peaks and valleys change within a single class. Cars are therefore a candidate for under-sampling, without the risk of losing information.

This observation led us to under-sample cars by $\frac{1}{2}$ in the preliminary classification tests. We need to assess that we could do this without loss of information.

We tried two under-sampling ratios, 1:2 and 1:4, and compared the results with the complete data set.

We have, so far, used 'automatic' weights which introduce class weights inversely
proportional to the class frequency. We are going to compare this with all class weights set to one, which we call 'manual'.

The cost factor C and kernel parameter gamma were tuned for each round of tests.

**RBF Kernel – One vs. Rest**

The following table shows the f1-score per class.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>f1-avg</th>
<th>f1-min</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>0.955</td>
<td>0.977</td>
<td>0.880</td>
<td>0.831</td>
<td>0.841</td>
<td>0.813</td>
<td>0.924</td>
<td>0.930</td>
<td>0.955</td>
<td>0.901</td>
<td>0.813</td>
</tr>
<tr>
<td>auto 1:2</td>
<td>0.947</td>
<td>0.976</td>
<td>0.879</td>
<td>0.827</td>
<td>0.844</td>
<td>0.807</td>
<td>0.921</td>
<td>0.934</td>
<td>0.956</td>
<td>0.899</td>
<td>0.807</td>
</tr>
<tr>
<td>auto 1:4</td>
<td>0.931</td>
<td>0.972</td>
<td>0.878</td>
<td>0.815</td>
<td>0.840</td>
<td>0.809</td>
<td>0.924</td>
<td>0.927</td>
<td>0.955</td>
<td>0.895</td>
<td>0.809</td>
</tr>
<tr>
<td>man</td>
<td>0.950</td>
<td>0.981</td>
<td>0.881</td>
<td>0.836</td>
<td>0.843</td>
<td>0.807</td>
<td>0.923</td>
<td>0.933</td>
<td>0.956</td>
<td>0.901</td>
<td>0.807</td>
</tr>
<tr>
<td>man 1:2</td>
<td>0.932</td>
<td>0.977</td>
<td>0.884</td>
<td>0.832</td>
<td>0.848</td>
<td>0.802</td>
<td>0.927</td>
<td>0.930</td>
<td>0.955</td>
<td>0.899</td>
<td>0.802</td>
</tr>
<tr>
<td>man 1:4</td>
<td>0.934</td>
<td>0.974</td>
<td>0.880</td>
<td>0.824</td>
<td>0.840</td>
<td>0.817</td>
<td>0.929</td>
<td>0.933</td>
<td>0.957</td>
<td>0.899</td>
<td>0.817</td>
</tr>
</tbody>
</table>

Plotting these figures directly is not legible as the difference between individual classes are much higher than the difference produced by under-sampling.

**RBF Kernel OvR- Random Undersampling**

We plot the difference of f1-score with the default choice of automatic weights without under-sampling. Scores are absolute values in the [0, 1] interval, not percentages.
There isn't a net overall winner. Under-sampling inevitably introduces losses. Automatic weights also appears to be a fair default choice for the RBF kernel.
Polynomial Kernel – One vs. One

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>f1-avg</th>
<th>f1-min</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>0.860</td>
<td>0.973</td>
<td>0.885</td>
<td>0.833</td>
<td>0.850</td>
<td>0.763</td>
<td>0.946</td>
<td>0.927</td>
<td>0.954</td>
<td>0.888</td>
<td>0.763</td>
</tr>
<tr>
<td>auto 1:2</td>
<td>0.871</td>
<td>0.973</td>
<td>0.886</td>
<td>0.832</td>
<td>0.852</td>
<td>0.762</td>
<td>0.945</td>
<td>0.929</td>
<td>0.956</td>
<td>0.889</td>
<td>0.762</td>
</tr>
<tr>
<td>auto 1:4</td>
<td>0.853</td>
<td>0.973</td>
<td>0.886</td>
<td>0.832</td>
<td>0.847</td>
<td>0.760</td>
<td>0.943</td>
<td>0.926</td>
<td>0.954</td>
<td>0.886</td>
<td>0.760</td>
</tr>
<tr>
<td>man</td>
<td>0.855</td>
<td>0.984</td>
<td>0.898</td>
<td>0.870</td>
<td>0.859</td>
<td>0.759</td>
<td>0.949</td>
<td>0.925</td>
<td>0.956</td>
<td>0.895</td>
<td>0.759</td>
</tr>
<tr>
<td>man 1:2</td>
<td>0.865</td>
<td>0.980</td>
<td>0.895</td>
<td>0.854</td>
<td>0.856</td>
<td>0.767</td>
<td>0.949</td>
<td>0.926</td>
<td>0.957</td>
<td>0.894</td>
<td>0.767</td>
</tr>
<tr>
<td>man 1:4</td>
<td>0.864</td>
<td>0.975</td>
<td>0.899</td>
<td>0.842</td>
<td>0.859</td>
<td>0.769</td>
<td>0.947</td>
<td>0.925</td>
<td>0.956</td>
<td>0.893</td>
<td>0.769</td>
</tr>
</tbody>
</table>

In this case we have a number of positive results. 1:2 under-sampling performs well, both with manual and automatic weights. We can see some information loss at a 1:4 ratio. The best results are obtained with manual weights and 1:2 under-sampling.

Polynomial Kernel – One vs. Rest

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>f1-avg</th>
<th>f1-min</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>0.765</td>
<td>0.978</td>
<td>0.871</td>
<td>0.832</td>
<td>0.825</td>
<td>0.743</td>
<td>0.928</td>
<td>0.916</td>
<td>0.952</td>
<td>0.868</td>
<td>0.743</td>
</tr>
<tr>
<td>auto 1:2</td>
<td>0.763</td>
<td>0.977</td>
<td>0.894</td>
<td>0.831</td>
<td>0.838</td>
<td>0.737</td>
<td>0.930</td>
<td>0.920</td>
<td>0.955</td>
<td>0.871</td>
<td>0.737</td>
</tr>
<tr>
<td>auto 1:4</td>
<td>0.743</td>
<td>0.974</td>
<td>0.893</td>
<td>0.821</td>
<td>0.835</td>
<td>0.736</td>
<td>0.926</td>
<td>0.918</td>
<td>0.954</td>
<td>0.867</td>
<td>0.736</td>
</tr>
<tr>
<td>man</td>
<td>0.831</td>
<td>0.982</td>
<td>0.881</td>
<td>0.856</td>
<td>0.830</td>
<td>0.778</td>
<td>0.938</td>
<td>0.917</td>
<td>0.954</td>
<td>0.885</td>
<td>0.778</td>
</tr>
<tr>
<td>man 1:2</td>
<td>0.829</td>
<td>0.979</td>
<td>0.884</td>
<td>0.847</td>
<td>0.838</td>
<td>0.784</td>
<td>0.930</td>
<td>0.917</td>
<td>0.953</td>
<td>0.885</td>
<td>0.784</td>
</tr>
<tr>
<td>man 1:4</td>
<td>0.871</td>
<td>0.977</td>
<td>0.880</td>
<td>0.844</td>
<td>0.834</td>
<td>0.776</td>
<td>0.929</td>
<td>0.913</td>
<td>0.952</td>
<td>0.886</td>
<td>0.776</td>
</tr>
</tbody>
</table>
Manual weights are providing a welcome improvement that brings the one-versus-rest method closer in performance to the one-versus-one. Under-sampling gives some positive results at 1:2 ratio for automatic weights and 1:4 for manual weights. Manual weights without under-sampling is the best all-round performer.

Conclusion:

• Under-sampling only yields net gains with the polynomial kernel in one-versus-rest.

• Ratios higher than 1:2 usually introduce some data loss and are to be avoided.

• Manual class weights significantly improve the performance of the polynomial kernel both in one-versus-one and one-versus-rest.
7.4.2 Oversampling

We are going to oversample the minority classes in two ways, first the 2 least represented classes (1 and 7), then the 4 least represented classes (1, 4, 6 and 7). The factors are chosen as follows:

In each case, we will test oversampling on a full dataset, and combine it with a 1:2 under-sampling of class 3. The C and gamma parameters are tuned in each instance using a grid search. There are 8 test cases in total, the cross-product of

\[ \{2 \text{ class}, 4 \text{ class}\} \times \{Id, 1:2\} \times \{auto, manual\} \]

<table>
<thead>
<tr>
<th>class</th>
<th>auto</th>
<th>Os x2 auto</th>
<th>Os x4 auto</th>
<th>Os x2 man</th>
<th>Os x4 man</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Bus or Coach</td>
<td>0.955</td>
<td>0.957</td>
<td>0.953</td>
<td>0.948</td>
<td>0.947</td>
</tr>
<tr>
<td>3 Car or light van</td>
<td>0.977</td>
<td>0.978</td>
<td>0.977</td>
<td>0.981</td>
<td>0.981</td>
</tr>
<tr>
<td>4 Car or light van plus trailer</td>
<td>0.880</td>
<td>0.881</td>
<td>0.881</td>
<td>0.882</td>
<td>0.882</td>
</tr>
<tr>
<td>5 Heavy van or minibus</td>
<td>0.831</td>
<td>0.833</td>
<td>0.831</td>
<td>0.832</td>
<td>0.832</td>
</tr>
<tr>
<td>6 Heavy van or minibus plus trailer</td>
<td>0.841</td>
<td>0.845</td>
<td>0.839</td>
<td>0.843</td>
<td>0.843</td>
</tr>
<tr>
<td>7 Special artic</td>
<td>0.813</td>
<td>0.804</td>
<td>0.816</td>
<td>0.809</td>
<td>0.808</td>
</tr>
<tr>
<td>8 Rigid</td>
<td>0.924</td>
<td>0.922</td>
<td>0.923</td>
<td>0.922</td>
<td>0.923</td>
</tr>
<tr>
<td>9 Rigid plus trailer</td>
<td>0.930</td>
<td>0.935</td>
<td>0.931</td>
<td>0.932</td>
<td>0.930</td>
</tr>
<tr>
<td>10 Artic</td>
<td>0.955</td>
<td>0.955</td>
<td>0.954</td>
<td>0.956</td>
<td>0.955</td>
</tr>
</tbody>
</table>

RBF kernel - One vs. Rest

We compare those results with the best results achieved previously with automatic weights.
The changes are less than 1 percent. Oversampling doesn't provide a net winner. Automatic weights without under or over-sampling remains the best performer.

**Polynomial Kernel – One vs. One**

We present the best performers compared with the best results previously achieved with manual weights:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>f1-avg</th>
<th>f1-min</th>
</tr>
</thead>
<tbody>
<tr>
<td>man</td>
<td>0.865</td>
<td>0.980</td>
<td>0.895</td>
<td>0.854</td>
<td>0.856</td>
<td>0.767</td>
<td>0.949</td>
<td>0.926</td>
<td>0.957</td>
<td>0.894</td>
<td>0.767</td>
</tr>
<tr>
<td>1:2 os 2 auto</td>
<td>0.871</td>
<td>0.974</td>
<td>0.886</td>
<td>0.837</td>
<td>0.852</td>
<td>0.769</td>
<td>0.948</td>
<td>0.926</td>
<td>0.955</td>
<td>0.891</td>
<td>0.769</td>
</tr>
<tr>
<td>os 2 man</td>
<td>0.863</td>
<td>0.984</td>
<td>0.895</td>
<td>0.868</td>
<td>0.857</td>
<td>0.766</td>
<td>0.949</td>
<td>0.923</td>
<td>0.956</td>
<td>0.896</td>
<td>0.766</td>
</tr>
<tr>
<td>1:2 os4 man</td>
<td>0.864</td>
<td>0.979</td>
<td>0.893</td>
<td>0.852</td>
<td>0.861</td>
<td>0.770</td>
<td>0.949</td>
<td>0.925</td>
<td>0.957</td>
<td>0.894</td>
<td>0.770</td>
</tr>
</tbody>
</table>
Oversampling classes 1 and 7, with manual weights, provides the best results. Paradoxically the improvement is in class 5 (Heavy Van) rather than in the targeted classes.

**Polynomial Kernel – One vs. Rest**

We present the best performers compared with the best results previously achieved with manual weights

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>f1-avg</th>
<th>f1-min</th>
</tr>
</thead>
<tbody>
<tr>
<td>man</td>
<td>0.831</td>
<td>0.982</td>
<td>0.881</td>
<td>0.856</td>
<td>0.830</td>
<td>0.778</td>
<td>0.938</td>
<td>0.917</td>
<td>0.954</td>
<td>0.885</td>
<td>0.778</td>
<td></td>
</tr>
<tr>
<td>1:4 man</td>
<td>0.871</td>
<td>0.977</td>
<td>0.880</td>
<td>0.844</td>
<td>0.834</td>
<td>0.776</td>
<td>0.929</td>
<td>0.913</td>
<td>0.952</td>
<td>0.886</td>
<td>0.776</td>
<td></td>
</tr>
<tr>
<td>os 2 man</td>
<td>0.864</td>
<td>0.982</td>
<td>0.881</td>
<td>0.855</td>
<td>0.831</td>
<td>0.749</td>
<td>0.939</td>
<td>0.914</td>
<td>0.953</td>
<td>0.885</td>
<td>0.749</td>
<td></td>
</tr>
<tr>
<td>1:2 os 4 man</td>
<td>0.865</td>
<td>0.979</td>
<td>0.883</td>
<td>0.848</td>
<td>0.839</td>
<td>0.749</td>
<td>0.934</td>
<td>0.912</td>
<td>0.953</td>
<td>0.885</td>
<td>0.749</td>
<td></td>
</tr>
</tbody>
</table>
Oversampling doesn't produce a net winner. Improvements in class 1 (Bus / Coach) are achieved at the expense of equal losses in class 7 (Special Artic.). So far the best performers are manual weights, or manual weights with a 1:4 under-sampling of class 3 (Car).

Conclusion:
Oversampling only produces a net gain for the polynomial kernel in one-versus-one. We summarize our best performers so-far in the following table:

<table>
<thead>
<tr>
<th>Weights</th>
<th>Under-sampling</th>
<th>Oversampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBF OvR</td>
<td>Auto</td>
<td>None</td>
</tr>
<tr>
<td>Poly OvO</td>
<td>Manual</td>
<td>None</td>
</tr>
<tr>
<td>Poly OvR</td>
<td>Manual</td>
<td>1:4 Class 3 or None</td>
</tr>
</tbody>
</table>
7.4.3 Synthetic Minority Over-sampling Technique (SMOTE)

The SMOTE algorithm is an oversampling technique that calculates synthetic samples by linear interpolation between the nearest neighbouring vectors. Our vectors are primarily composed of signal waveforms, in order to evaluate how they respond to the algorithm, we wrote a function to plot the synthetic vectors for a given class in our dataset. This tool also helped us estimate the number of nearest-neighbours most likely to perform well.

We also conducted spot tests, looking at the classification report accuracy and recall values, to see how the classifiers responded with different parameters of the algorithm (oversampling ratio and kNN). With low ratios and kNN=1, we could see no difference in the number of support vectors in the oversampled classes. With kNN larger than 3 we could see too much noise being introduced in the data.

We show here a plot of synthetic vectors generated by SMOTE for a Bus. The plot is produced with a 128 samples waveform, our vectors only have 32 samples.
We tested three settings for the SMOTE algorithm, affecting class 1 (Bus / Coach) and class 7 (Special Artic.)

<table>
<thead>
<tr>
<th></th>
<th>Class 1</th>
<th></th>
<th>Class 7</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ratio</td>
<td>kNN</td>
<td>ratio</td>
<td>kNN</td>
</tr>
<tr>
<td>Smote 1</td>
<td>32</td>
<td>2</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Smote 2</td>
<td>24</td>
<td>2</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>Smote 3</td>
<td>32</td>
<td>2</td>
<td>16</td>
<td>1</td>
</tr>
</tbody>
</table>

We present the results in two charts: a comparison with the best previously found method, and the difference between SMOTE and oversampling of the same classes.

**RBF kernel - One vs. Rest**

We refer the reader to the spreadsheet `datasetC_rbf_ovr_report.ods` in the CD-ROM for numerical data.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>f1-avg</th>
<th>f1-min</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>0.955</td>
<td>0.977</td>
<td>0.880</td>
<td>0.831</td>
<td>0.841</td>
<td>0.813</td>
<td>0.924</td>
<td>0.930</td>
<td>0.955</td>
<td>0.901</td>
<td>0.813</td>
</tr>
<tr>
<td>Smote 1 auto</td>
<td>0.960</td>
<td>0.978</td>
<td>0.884</td>
<td>0.832</td>
<td>0.842</td>
<td>0.800</td>
<td>0.923</td>
<td>0.933</td>
<td>0.955</td>
<td>0.901</td>
<td>0.800</td>
</tr>
<tr>
<td>Smote 2 auto</td>
<td>0.969</td>
<td>0.978</td>
<td>0.882</td>
<td>0.833</td>
<td>0.841</td>
<td>0.795</td>
<td>0.921</td>
<td>0.934</td>
<td>0.955</td>
<td>0.901</td>
<td>0.795</td>
</tr>
<tr>
<td>Smote 1 man</td>
<td>0.942</td>
<td>0.981</td>
<td>0.880</td>
<td>0.838</td>
<td>0.842</td>
<td>0.799</td>
<td>0.922</td>
<td>0.932</td>
<td>0.955</td>
<td>0.899</td>
<td>0.799</td>
</tr>
<tr>
<td>Smote 2 man</td>
<td>0.960</td>
<td>0.981</td>
<td>0.882</td>
<td>0.838</td>
<td>0.843</td>
<td>0.787</td>
<td>0.920</td>
<td>0.932</td>
<td>0.956</td>
<td>0.900</td>
<td>0.787</td>
</tr>
</tbody>
</table>

**RBF Kernel OvR - SMOTE**

delta f1-score. Baseline: automatic class weights
SMOTE doesn't produce a net benefit. SMOTE performs better than oversampling in class 1 and worse in class 7.

**Polynomial Kernel – One vs. One**

We refer the reader to the spreadsheet datasetC_poly_ovo_report.ods in the CD-ROM for numerical data.

**RBF Kernel OvR - SMOTE vs Oversampling**

delta f1-score. Baseline: 2 classes oversampling

**Poly Kernel OvO - SMOTE**

delta f1-score. Baseline: 2 classes oversampling, class weights = 1 (manual)
SMOTE doesn't produce a net benefit. SMOTE performs better than oversampling in class 1 and worse in class 7.

**Polynomial Kernel – One vs. Rest**

We refer the reader to the spreadsheet datasetC_poly_ovr_report.ods in the CD-ROM for numerical data.
SMOTE with manual weights provides an improvement, however manual weights with no oversampling remains the best performer.

**Conclusion:** SMOTE performs better than oversampling for class 1 and worse for class 7. On the whole, the SMOTE algorithm did not produce net gains. These results require an interpretation that will be the topic of the next chapter.
7.5 Noise and Outliers

The previous results exhibit problems with the class 7 Special Artic category. Class 7 is not only the worst performer of all classes, it seems to respond adversely to class imbalance techniques.

We know that Support Vector Machines are sensitive to noise and outliers. This is also true for SMOTE.

“It has been well studied in the literature that SVMs are also sensitive to the noise and outliers present in datasets. Therefore, it can be argued that although the existing class imbalance learning methods can make the SVM algorithm less sensitive to the class imbalance problem, it can still be sensitive to noise and outliers present in datasets, which could still result in suboptimal models. In fact, some class imbalance learning methods, such as random oversampling and SMOTE, can make the problem worse by duplicating the existing outliers and noisy examples or introducing new ones.” (Batuwita & Paladey, 2013)

There are only 15 samples in class 7, it is therefore worth checking their waveform plots and the video capture provided by Clearview. PT26M22.2S waveform stands out from class 7 and appears similar to a vehicle with trailer of class 4 or 6.
The corresponding video capture is out of focus.

We believe this is a class 6 vehicle, similar to the following example.

We therefore constructed a new set of XML files, moved PT26M22.2S to class 6 and ran the previous tests with the new metadata.
7.6 Results with New Dataset

This table clarifies the names of the datasets we are using. The naming convention is used throughout the project for XML, batch files, logs and reports.

<table>
<thead>
<tr>
<th>Name</th>
<th>Dataset Properties</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Original XML file</td>
<td>Not used</td>
</tr>
<tr>
<td>B</td>
<td>Corrections from Clearview Note on video and data</td>
<td>Prototyping</td>
</tr>
<tr>
<td>C500</td>
<td>Bus + Trailer ignored, cars limited to 500 samples</td>
<td>First SVM and AdaBoost tests</td>
</tr>
<tr>
<td>C</td>
<td>Bus + Trailer ignored</td>
<td>Previous class-imbalance tests</td>
</tr>
<tr>
<td>D</td>
<td>Bus + Trailer ignored, PT26M22.2S class 6</td>
<td>This chapter</td>
</tr>
</tbody>
</table>

We are presenting the best performers for each Kernel, compared with a baseline obtained with either automatic or manual weights. We also include the best performing method obtained with dataset C in the previous chapters.

RBF Kernel - One vs. Rest

![Dataset D - RBF Kernel OvR](image-url)

delta f1-score. Baseline: automatic class weights
Automatic class weights is a fair default choice. Oversampling classes 1, 4, 6, 7 reduces the performance gap between classes. There is a net gain over the previous dataset.

**Polynomial Kernel – One vs. One**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>f1-avg</th>
<th>f1-min</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>0.964</td>
<td>0.978</td>
<td>0.881</td>
<td>0.834</td>
<td>0.858</td>
<td>0.837</td>
<td>0.922</td>
<td>0.936</td>
<td>0.955</td>
<td>0.907</td>
<td>0.834</td>
</tr>
<tr>
<td>Osamp 4 auto</td>
<td><strong>0.960</strong></td>
<td><strong>0.978</strong></td>
<td><strong>0.884</strong></td>
<td><strong>0.834</strong></td>
<td><strong>0.857</strong></td>
<td><strong>0.844</strong></td>
<td><strong>0.924</strong></td>
<td><strong>0.936</strong></td>
<td><strong>0.955</strong></td>
<td><strong>0.908</strong></td>
<td><strong>0.834</strong></td>
</tr>
<tr>
<td>Data C auto</td>
<td>0.955</td>
<td>0.977</td>
<td>0.880</td>
<td>0.831</td>
<td>0.841</td>
<td>0.813</td>
<td>0.924</td>
<td>0.930</td>
<td>0.955</td>
<td>0.901</td>
<td>0.813</td>
</tr>
</tbody>
</table>

The best results are obtained with class weights = 1, under-sampling class 3 by 1:2 and oversampling classes 1 and 7 with SMOTE.
Polynomial Kernel – One vs. Rest

The best results are obtained with the same parameters as for the One vs. One method.

Vehicle Classification from Inductive Loop Signature
7.7 Conclusion

We present our best performers, obtained with dataset D, using Stratified Shuffle Split with 1000 iterations, a training set of 80% and test set 20%.

The combination of SMOTE and under-sampling improves the results obtained with the Polynomial Kernel, in particular in One-versus-Rest. This is useful for reasons of computational cost.

The improvements with the Gaussian Kernel are due to the fact that SMOTE helped us identify outliers, and remove them from the training data.

7.7.1 Numerical results for the Gaussian Kernel

In the case of the Gaussian Kernel, our detailed results are as follows. Note that the averages in this table are weighted by the class frequencies, we also show the unweighted average used previously.
### Support Vector Machine, Gaussian Kernel, One vs. Rest

**Stratified Shuffle Split, training 80%, test 20%, 1000 iterations**

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus</td>
<td>96%</td>
<td>81%</td>
<td>88%</td>
<td>1000</td>
</tr>
<tr>
<td>Car</td>
<td>99%</td>
<td>96%</td>
<td>98%</td>
<td>220000</td>
</tr>
<tr>
<td>Car + Trailer</td>
<td>86%</td>
<td>91%</td>
<td>89%</td>
<td>8000</td>
</tr>
<tr>
<td>Heavy van</td>
<td>76%</td>
<td>93%</td>
<td>83%</td>
<td>32000</td>
</tr>
<tr>
<td>Heavy van + Trailer</td>
<td>87%</td>
<td>82%</td>
<td>84%</td>
<td>11000</td>
</tr>
<tr>
<td>Special Artic</td>
<td>92%</td>
<td>76%</td>
<td>83%</td>
<td>3000</td>
</tr>
<tr>
<td>Rigid</td>
<td>93%</td>
<td>93%</td>
<td>93%</td>
<td>29000</td>
</tr>
<tr>
<td>Rigid + Trailer</td>
<td>94%</td>
<td>92%</td>
<td>93%</td>
<td>25000</td>
</tr>
<tr>
<td>Articulated</td>
<td>95%</td>
<td>96%</td>
<td>95%</td>
<td>64000</td>
</tr>
<tr>
<td><strong>Average weighted</strong></td>
<td>95%</td>
<td>95%</td>
<td>95%</td>
<td>393000</td>
</tr>
<tr>
<td><strong>Average unweighted</strong></td>
<td>91%</td>
<td>89%</td>
<td>90%</td>
<td></td>
</tr>
</tbody>
</table>

We show more detailed results in the following confusion matrix:

<table>
<thead>
<tr>
<th></th>
<th>Bus</th>
<th>Car</th>
<th>Car + T</th>
<th>HV</th>
<th>HV + T</th>
<th>Sp Art</th>
<th>Rigid</th>
<th>Rig + T</th>
<th>Artic.</th>
<th>recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus</td>
<td>805</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>191</td>
<td>0</td>
<td>0</td>
<td>80.5%</td>
</tr>
<tr>
<td>Car</td>
<td>0</td>
<td>21119</td>
<td>0</td>
<td>8702</td>
<td>0</td>
<td>7</td>
<td>95</td>
<td>87</td>
<td>96.0%</td>
<td></td>
</tr>
<tr>
<td>Car + T</td>
<td>0</td>
<td>0</td>
<td>7285</td>
<td>2</td>
<td>710</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>91.1%</td>
</tr>
<tr>
<td>Heavy van</td>
<td>0</td>
<td>1586</td>
<td>0</td>
<td>29684</td>
<td>190</td>
<td>0</td>
<td>540</td>
<td>0</td>
<td>0</td>
<td>92.8%</td>
</tr>
<tr>
<td>HV + T</td>
<td>0</td>
<td>0</td>
<td>1125</td>
<td>335</td>
<td>9021</td>
<td>26</td>
<td>0</td>
<td>295</td>
<td>198</td>
<td>82.0%</td>
</tr>
<tr>
<td>Spec. Artic</td>
<td>0</td>
<td>0</td>
<td>33</td>
<td>2</td>
<td>116</td>
<td>2266</td>
<td>216</td>
<td>0</td>
<td>367</td>
<td>75.5%</td>
</tr>
<tr>
<td>Rigid</td>
<td>31</td>
<td>35</td>
<td>0</td>
<td>406</td>
<td>0</td>
<td>69</td>
<td>27045</td>
<td>0</td>
<td>1414</td>
<td>93.3%</td>
</tr>
<tr>
<td>Rigid + T</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>371</td>
<td>0</td>
<td>0</td>
<td>23123</td>
<td>1506</td>
<td>92.5%</td>
</tr>
<tr>
<td>Artic.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>97</td>
<td>1221</td>
<td>1087</td>
<td>61593</td>
<td>96.2%</td>
</tr>
</tbody>
</table>

**precision** 96.3% 99.2% 86.3% 75.8% 86.7% 92.2% 92.6% 94.0% 94.5%

#### 7.7.2 Comparison with published results

In the case of the Gaussian kernel, we want to compare our results with those published amongst our literature review. This isn't always possible as some authors don't publish numerical results, or the number of categories is too small. Thankfully two authors publish their results in a confusion matrix, which indicates that they are reporting recall rates. (Meta & Cinsdikici, 2010) (Ki & Baik, 2006) We assumed that this is also the case for Feng & Minzhe (2009) who use the term “classification correct” for their metric.
These authors classify into 4 or 5 classes of vehicles: Bus, Car, Van and Truck, the fifth class being Motorcycles. We mapped our categories as follows.

<table>
<thead>
<tr>
<th>SWISS 10</th>
<th>Literature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus</td>
<td>Bus</td>
</tr>
<tr>
<td>Car</td>
<td>Car</td>
</tr>
<tr>
<td>Car + Trailer</td>
<td></td>
</tr>
<tr>
<td>Heavy van</td>
<td>Van</td>
</tr>
<tr>
<td>Heavy van + Trailer</td>
<td></td>
</tr>
<tr>
<td>Special Artic Trucks</td>
<td></td>
</tr>
<tr>
<td>Rigid</td>
<td></td>
</tr>
<tr>
<td>Rigid + Trailer</td>
<td></td>
</tr>
<tr>
<td>Articulated</td>
<td></td>
</tr>
</tbody>
</table>

For each of the categories from the literature, we calculated a weighted average of the recall rate obtained with our method.

<table>
<thead>
<tr>
<th>Our results</th>
<th>Meta &amp; Cinsdikici</th>
<th>Feng &amp; Mingzhe</th>
<th>Ki &amp; Baik</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>2 ILD</td>
<td>2 ILD</td>
<td>AMR</td>
</tr>
<tr>
<td>Algorithm</td>
<td>SVM</td>
<td>PCA + BPNN</td>
<td>SVM</td>
</tr>
<tr>
<td>recall</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bus</td>
<td>81.0%</td>
<td>100.0%</td>
<td>80.0%</td>
</tr>
<tr>
<td>Car</td>
<td>95.8%</td>
<td>98.0%</td>
<td>89.0%</td>
</tr>
<tr>
<td>Van</td>
<td>90.2%</td>
<td>89.7%</td>
<td>81.0%</td>
</tr>
<tr>
<td>Trucks</td>
<td>94.0%</td>
<td>92.8%</td>
<td>86.0%</td>
</tr>
<tr>
<td>Average</td>
<td>90.3%</td>
<td>95.1%</td>
<td>84.0%</td>
</tr>
</tbody>
</table>

This number of classes was not the target of our study, however our results perform honourably.

7.7.3 Comparison with Clearview Traffic Algorithm

Clearview Traffic were kind enough to provide us with the results obtained from their classifier. Their current algorithm is a tree-based classifier based on the detection of peak and valleys in the signal.

In order to draw a fair comparison, we ran our classifier on a dataset that includes the
Bus + Trailer sample. Those results differ slightly from the previous dataset in the Bus category.

<table>
<thead>
<tr>
<th></th>
<th>Clearview algorithm</th>
<th>SVM Gaussian Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>precision</td>
<td>recall</td>
</tr>
<tr>
<td>Bus</td>
<td>50%</td>
<td>83%</td>
</tr>
<tr>
<td>Car</td>
<td>95%</td>
<td>99%</td>
</tr>
<tr>
<td>Car + Trailer</td>
<td>65%</td>
<td>95%</td>
</tr>
<tr>
<td>Heavy van</td>
<td>86%</td>
<td>61%</td>
</tr>
<tr>
<td>Heavy van + Trailer</td>
<td>81%</td>
<td>74%</td>
</tr>
<tr>
<td>Special Artic</td>
<td>8%</td>
<td>29%</td>
</tr>
<tr>
<td>Rigid</td>
<td>86%</td>
<td>84%</td>
</tr>
<tr>
<td>Rigid + Trailer</td>
<td>61%</td>
<td>19%</td>
</tr>
<tr>
<td>Articulated</td>
<td>69%</td>
<td>75%</td>
</tr>
<tr>
<td>Avg weighted</td>
<td>85%</td>
<td>84%</td>
</tr>
<tr>
<td>Avg unweighted</td>
<td>67%</td>
<td>69%</td>
</tr>
</tbody>
</table>

In this case, our method clearly outperforms the previous algorithm.
8 Estimation of Computational Cost

8.1 Relevance of computational cost

Vehicle classification is performed on low-power embedded systems, housed in a cabinet on the side of the road. To take our solution from the laboratory to an industrial application, we need to evaluate how the pre-processing and classifier algorithms would run on an embedded system. In fact, an estimate of the computational cost of our solution is one of the requirements from our client.

A typical platform in use by our client is based an ARMv4 architecture, clocked at 8MHz, which delivers around 10 MIPS. Power consumption is a decisive criteria in the design of such equipment. We will therefore evaluate the computation time on Classic ARM, and on a more modern Cortex-M4 architecture, at 8MHz clock speed.

These type of processors don't offer hardware floating point units. We will therefore produce a prototype implementation in fixed-point arithmetic. This is a requirement, as the computation time can only be derived from a fixed point implementation.

8.2 High Level Overview

We will estimate the computational complexity of the decision function for Support Vector Machines with polynomial or RBF kernel.

Only the pre-processing and the decision function are to be implemented on the embedded system. The training stage is done in a high-level language, with floating point arithmetic. The support vectors, the values for $\alpha_i$, $b$ and kernel parameters are outputs from the high-level implementation, and need to be scaled to fixed-point representation for the embedded classifier.

The decision function, for two classes, can be expressed as

$$f(x) = \sum_i \alpha_i y_i K(x_i, x) + b$$

where $x_i$ are the support vectors, $y_i \in \{-1, 1\}$ represent class membership, $\alpha_i$ and
b are calculated at the training stage. (Bishop, 2006)

The kernel functions are, for the polynomial kernel: \( K(x, x') = (\gamma \langle x | x' \rangle + c)^3 \)
for the Gaussian kernel: \( K(x, x') = \exp(-\gamma |x - x'|^2) \)

The parameters that govern the CPU cycle cost are:

1. Number of dimensions for the input and support vectors
2. Total number of classifiers. Dependent on the number of support vectors and the multi-class method.
3. CPU cycle cost of multiplications. On ARM architecture, dependent on the number of significant bytes of the multiplicand.
4. CPU cycle cost of exponential function. Dependent on (3) and on the precision required.

If \( d \) is the number of dimensions of the support vectors, \( M \) the cost of multiplication, the computation of the kernel function is \( O(d \times M) \). If \( c \) is the number of classifiers, the computation of the decision function is \( O(c \times d \times M) \).

For a multi-class decision, we need to evaluate the total number of Kernel function calls involved by the One-versus-one method. This is the object of the next section.

### 8.3 Number of support vectors in One-vs-One.

We know that the number of classifiers in One-vs-One, for \( n \) classes is \( 1/2 n (n-1) \).

We want to calculate the total number of calls to the Kernel function, thus the total number of support vectors involved.

We have \( n \) classes and we call \( s_i \) the number of support vectors for class \( i \).

When we compare class 1 with the \( n-1 \) other classes, the number of support vectors
required is \( (n-1)s_i + \sum_{i=2}^{n} s_i \) which we can re-write \( (n-2)s_i + \sum_{i=1}^{n} s_i \)

When we compare class \( k \) with the other classes, for \( k \in [2, n-1] \) the number of support vectors is

\[
(n-1)s_k + \sum_{i=1}^{k-1} s_i + \sum_{i=k+1}^{n} s_i
\]

which we can also re-write \( (n-2)s_k + \sum_{i=1}^{n} s_i \)

The reader will easily find that the case \( k=n \) is similar to \( k=1 \).

If we sum these results for \( k \in [1, n] \) we get

\[
(n-2) \sum_{i=1}^{n} s_i + n \sum_{i=1}^{n} s_i = (2n-2) \sum_{i=1}^{n} s_i
\]

We have however counted each classifier twice, once for \( s_i \) vs. \( s_j \) and again for \( s_j \) vs. \( s_i \)

Therefore the total count is \( (n-1) \sum_{i=1}^{n} s_i \) or \( n(n-1)\bar{s} \) where \( \bar{s} \) is the average number of support vector per class.

**Conclusion:** If we have \( n \) classes and we call \( s_i \) the number of support vectors for class \( i \), the one-versus-one method requires \( (n-1) \sum_{i=1}^{n} s_i \) Kernel function calls.
In our case, for our best performing classifiers, inclusive of class imbalance corrections:

<table>
<thead>
<tr>
<th>Multi-class</th>
<th>Number of SV per class / classifier</th>
<th>Multiplier</th>
<th>Total SV</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM RBF</td>
<td>30, 149, 73, 107, 85, 49, 106, 127, 156</td>
<td>1</td>
<td>882</td>
</tr>
<tr>
<td>SVM Polynomial</td>
<td>12, 34, 18, 42, 24, 30, 34, 63</td>
<td>8</td>
<td>2248</td>
</tr>
<tr>
<td>SVM Polynomial</td>
<td>16, 59, 51, 231, 84, 55, 120, 70, 106</td>
<td>1</td>
<td>792</td>
</tr>
</tbody>
</table>

Although the One-versus-Rest method requires more support vectors for each classifier, it retains the advantage in the total number of operations.

This result is computed from our dataset, which ignored the motorcycle category. In practice, the final system will have one more class, and Clearview would collect more data in the less-represented categories. This will result in a higher number of support vectors.

### 8.4 Fixed point implementation and required resolution

The type of CPUs used in our target system don't offer a floating point unit. Therefore, the pre-processing and classification are to be implemented in fixed-point arithmetic. This is a common scenario in embedded development.

We are going to implement the decision function for a Support Vector Machine, with a polynomial kernel, for two classes only. This is an incomplete implementation, but it provides enough opportunity for a first assessment of fixed-point arithmetic.

Both the Kernel and the decision functions use multiply-accumulate operations (ARM acronym SMLA, SMLAL). The cost of these operations depends the number of registers used for the accumulator. With 32 bits accumulators, operands need to be scaled down to prevent overflow, at the loss of precision. ARM offers a Signed Multiply-Accumulate Long instruction, which multiplies 32bit inputs into a 64bits accumulator.

We also need to assess the range of the variables used for computation, in order to choose the fixed-point representation of our variables.

Our workflow is therefore:
1. Implement the kernel and decision functions in IEEE 754 floating point with a log of each variable range.

2. Compute the required fixed point range and delta for each operation.

3. Implement in fixed-point with 32 and 64 bits accumulators.

4. Compare the output precision with the floating point implementation.

We conduct this analysis for the polynomial kernel only. The functions pseudo-code is:

```plaintext
(* Polynomial Kernel function degree 3 *)
1 function Kernel(x, sv, gamma)
2     dp ← dot_product(x, sv)
3     gamma_dp ← gamma * dp
4     square ← gamma_dp * gamma_dp
5 return square * gamma_dp

(* Support Vector Machine decision function *)
6 function Decision(x, sv[], alpha[], gamma, intercept)
7     mla ← 0
8     for i ← 0 to length(sv)
9         kf ← Kernel(x, sv[i], gamma)
10        mla ← mla + alpha[i] * kf
11     endfor
12 return intercept – mla
```

We determined the range of the variables for each vehicle class, and for a fixed choice of parameters: C=5, gamma=0.5. We then retained the maximum range value for all classes. This allows us to complete the two range columns in the following table.

We are using the Texas Instruments Q number format to represent fixed point numbers. Q_m.n indicates m bits for the integer part and n bits for the fractional part. We are using signed integers, m+n+1 = 32 or 64 bits. The range is determined by m and the resolution by n.
We then populate the table ensuring that:

1. The integer bits is larger than the variable range. For example, line 4, Q9.54 has 9 integer bits, 7 bits are required for the range.

2. The fractional bits add up for multiplications. Example, line 5: 27 + 24 = 51

3. The multiplier and multiplicand are both expressed on 32 bits.

The table is populated from output to inputs. For each computation, the range of the output is determined first, then the fractional bits of the inputs is determined to prevent overflow. The Q values are dependant on the choice of C and gamma.

The 32 bits implementation failed to provide sufficient resolution for the decision function. There are two reasons for this: (1) 32 bit accumulators reduce the resolution of the operands below 16 bits to prevent overflow. (2) The decision function suffers from cancellation effect. The average output amplitude for the decision function is around 5.0 to 10, whilst the range required for the accumulator is around $10^{5}$ ($2^{18}$).

On the other hand, with 64 bits accumulators, the error for the decision function is $10^{-4}$ average, $2.10^{-4}$ peak. The choice of fixed-point representation Q values are indicated in the two last columns of the table above.

Conclusion: Fixed-point implementation of the support vector machine decision function yields sufficient resolution when using 32 bits registers for variables and 64 bits for multiplications and multiply-accumulate registers.
8.5 Number of Operations and CPU cycle count

Our aim is to achieve an estimate of CPU cycle count for a single classifier. The best approach is to implement the classifier in C or assembly language and run the code on a development board. This is however beyond the scope of this dissertation. We will limit ourselves to estimate the number of assembly instructions by derivation. This estimate is produced from personal experience of decompiling on Intel x86 architecture and study of the ARM architecture and ALU properties.

The kernel functions and the decision functions are simple enough to estimate their instruction count in assembly language. We used “Instruction Cycle Timings” (Sloss & Wright. 2013) and the “Cortex-M4 Technical Reference Manual” (ARM, 2014) to derive a CPU cycle count from the assembly instructions.

Clearview’s current platform is the ARM7TDMI, with the ARMv4 instruction set, which is called “Classic ARM” in the literature. The cost of multiplications on Classic ARM depends on the number of significant bytes of the multiplier, whereas it is performed in a single cycle on Cortex-M4 architecture. (ARM, 2014) We produced two separate counts, for Classic ARM and for Cortex-M4.

The code is built form 3 subroutines: 1. Kernel computation, 2. SVM decision function, 3. Multiclass decision with voting. The spreadsheet allows to input the following parameters: vector size (number of dimensions), average number of support vector per class, number of classes, CPU cycle count of multiplications, number of terms in the Taylor expansion of the exponential function.

We show here an extract of the spreadsheet which can be found on the CD-ROM. This is for the Polynomial Kernel, using 64 bits signed multiply long and multiply-accumulate long instructions.

The estimate is produced for 10 vehicle classes and a more realistic number of support vectors than in our current dataset. We extrapolated the number of support vectors from the values in well-represented classes (e.g. cars, articulated lorry).
For the RBF Kernel, we assumed the exponential would be computed with a lookup table and a Taylor series expansion. With this technique, the Taylor series is applied to a value smaller than 0.5. This converges rapidly and yields to at least one bit of resolution per term. We allowed for 22 terms in the Taylor series approximation of the exponential. This value is derived from the resolution used for the fixed-point polynomial
computation.

Our estimates are summarised in the following table, in CPU cycle counts, and in milliseconds for a clock speed of 8MHz.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Polynomial Kernel</th>
<th>RBF Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Classic ARM (8MHz)</strong></td>
<td>kCYC 536 ms 70</td>
<td>kCYC 942 ms 120</td>
</tr>
<tr>
<td>Cortex-M4</td>
<td>kCYC 275 ms 35</td>
<td>kCYC 548 ms 70</td>
</tr>
</tbody>
</table>

The pre-processing produces a very small instruction count: around 22 kCYC (22000 Instruction cycles), equivalent to 3ms at 8MHz.

Conclusion: Fixed-point implementation of a Support Vector Machine classifier on Classic ARM architecture appears to yield realistic computation times. This estimate is not derived from an executable implementation, ignores pipeline and memory access issues. It is therefore optimistic, but likely to fall within one order of magnitude.
9 Conclusion

9.1 Summary of achievements

We have compared the performance of machine learning algorithms in order to classify vehicles in a 10 category scheme, called the SWISS-10. In this comparison, Support Vector Machines gave better results than Adaptive Boosting, the best results being obtained with a Gaussian Kernel.

When reduced to four categories, our results compare honourably with results published in the literature. On nine categories (motorcycles excluded), our results outperformed the algorithm used by Clearview Traffic.

We have shown that a combination of class-imbalance techniques, under-sampling and Synthetic Minority Oversampling Technique (SMOTE) can significantly improve the results obtained with a Polynomial kernel. We have also seen that, as a side-effect, the SMOTE algorithm will flag the presence of outliers in the data.

We have estimated that the computation time of an SVM classifier with a Gaussian Kernel on ARMv4 architecture at 8Mhz is in the order of 130ms. Although this estimate is obtained from derivation rather than implementation in ARM Assembly Language, it indicates the potential for running the classifier on an embedded platform such as the one currently used by our client, and on a Cortex-M4 platform.

We have also shown that the classifier can be implemented in fixed-point arithmetic, with sufficient resolution, provided the multiply-accumulate registers are 64 bits wide.

9.2 Problems encountered an lessons learnt

The main problems encountered were related to the development of class imbalance techniques. The first prototype appear to deliver counter-productive results, this was due to unexpected outliers in the data. Dealing with the unexpected, and understanding why certain algorithms behave in a counter-intuitive way, required further reading and advice from my supervisor.

Another issue was the wrong estimate of the software development effort and of the
time required to run the experiments. This didn't cause problems thanks to a large amount of slack factored into the project plan.

In our case, part of the development was unforeseen at the specification stage, namely class-imbalance techniques, and the time required to run the experiments was not properly accounted at the planning stage. The lesson learnt is that estimating the development effort is difficult, and that sufficient slack needs to be factored into the project plan.

The amount of unit tests and the coverage of the test harness falls below the standards expected for commercial code or open source software libraries. The lesson learnt is two folds: (1) unit tests should be written alongside code development, (2) testing accounts for a significant part of the development effort, which needs to be factored in.

9.3 Evaluation of deliverables and software implementation

On the whole, the Python language, the functional programming approach and the Numpy, SciPy and Scikit-learn libraries offered a rather painless development environment.

The use of a relational database, included in the specification, didn't appear to be required. We used built-in serialization to store and speed-up access to numerical data, whilst meta-data and user data were represented in structured formats such as XML and JSON.

The main limitation of the current implementation is that it is single-threaded. If it were to be used regularly for parameter tuning, a multi-threaded implementation would be beneficial.

9.4 Social, Legal, Ethical and Professional Issues

9.4.1 Open Source Licensing in commercial applications.

The main legal issue in this project is the use of Open Source software within a commercial product. This is clearly allowed, even under the more restrictive GNU Public Licence (GPL). The main consequences for Clearview Traffic is essentially a
duty of displaying copyright notices in their code and possibly in their literature. The code developed for the dissertation is not Open Source, but it relies on Open Source language(s) and libraries. Should Clearview Traffic use parts of the code developed for this dissertation, they will be in a legal position similar to any other manufacturer using Open Source software in their products.

Python is released under its own Python Software Foundation (PSF) Licence, which allows commercial use. Scikit-learn is released under the BSD license. SciPy and NumPy are released under their own license that is compatible with the BSD and MIT licences. The LIBSVM library is released under its own license that is a modified BSD license. LIBSVM can be used in commercial products provided that the use of the library is indicated and that the Libsvm Copyright file is retained in the software.

Licence compatibility isn't an issue, as it is unlikely that Clearview Traffic will want to release the source code related to their products.

9.4.2 Social and Ethical issues

Road traffic monitoring raises few ethical and social issues. Induction loop hardware is used to gather statistical data or to assist automated payment. It is hard to imagine how the data collected can impinge on privacy. The data collected is essentially of statistical nature, individual vehicle signatures are discarded. The video capture provided to us was only collected for the purpose of product development.

On a normally running system, identifying a specific vehicle is not possible. As far as I know, the vehicle speed measurement is discarded and would not be accurate enough to be used for law enforcement.

Road traffic monitoring doesn't promote any particular use of motor vehicles. As it helps achieve better planning and maintenance of the road infrastructure, the effect is rather beneficial both for the taxpayer and society.

There would be ethical issues with classification errors in the case of an automated payment system. This falls under the regulatory framework from government and local authorities.

9.4.3 Professional duty of confidentiality

I have a duty to keep a degree of confidentiality around the project, in essence to
prevent Clearview Traffic’s competitors to gather information about this project. My code repository isn’t public. I access the repository using git over SSH with key authentication, web access is controlled by a strong password. My development machine, and the machine used to run experiments, are running Debian Linux and benefit from common-sense network security measures.

On a social engineering side, I abstain from posting about the project online. My LinkedIn page mentions that I am using Python and machine-learning, but there is no mention of vehicle classification.

### 9.4.4 Environmental issues

The power consumption of roadside equipment is, by far, the largest environmental concern of this type of system. Current systems run on low power embedded platforms, and can be solar powered. Even if our solution requires a hardware upgrade, power consumption is one of the main design drivers in embedded solutions, and manufacturers compete directly to offer the best processing power per watt. In fact, embedded processor power ratings are usually expressed in milliwatts per MHz of clock speed.

Other environmental issues such as waste management are mitigated by the fact that we are dealing with small volumes of devices compared with consumer products, and that the equipment is managed and maintained by professionals during its entire life cycle.

### 9.5 Work plan and project management

The project plan for this dissertation differed significantly from that of a full-time student due to personal circumstances. As a part-time student, I happen to have completed all modules and exams by the end of Semester 1, 2013. I also had obligations to accept part-time contract work alongside the dissertation.

The first meeting with our client, Clearview Traffic, was held on 10th January 2014. After an initial period of research and prototyping, a first project plan was drafted mid-February, with a potential finish date planned for mid-June and an end of software development planned for end-April.

By the writing of the Interim report in May, difficulties had been encountered with class...
imbalance, implementation and verification of the SMOTE algorithm, and about a month delay had occurred with software development. The time required to conduct collect experimental data with AdaBoost had also been underestimated. A new project plan was drawn for the release of the Interim report, with a finish date planned mid-August and an end of development end-June. This plan proved accurate so far.

A risk evaluation was carried out at the time of the dissertation proposal. As discussed previously, the main risk encountered was a wrong estimation of the development effort and occasional difficulty with the implementation.

**Technical risks**

<table>
<thead>
<tr>
<th>risk description</th>
<th>probability</th>
<th>severity</th>
<th>Contingency / Our solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical hardware failure</td>
<td>2</td>
<td>3</td>
<td>Use redundant hardware. Image system drives. Document software configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>We have 2 Debian desktop machines, 1 Win7 laptop.</em></td>
</tr>
<tr>
<td>Loss of data</td>
<td>1</td>
<td>5</td>
<td>Backup to external drive and internet. Use version control with online repository.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Include data and reports to the repository.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>500GB external drive backup. We use git with Bitbucket (online repository).</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>Practice sensible network security.</em></td>
</tr>
<tr>
<td>Difficulty with implementation</td>
<td>3</td>
<td>3</td>
<td>Use well-documented software library. Seek advice from supervisor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>Python, SciPy, Scikit-learn have excellent documentation.</em></td>
</tr>
<tr>
<td>Lack of support or documentation for software library</td>
<td>2</td>
<td>3</td>
<td>Use widely used and well-documented libraries.</td>
</tr>
<tr>
<td>Wrong estimation of development effort.</td>
<td>4</td>
<td>4</td>
<td>Plan slack in development tasks.</td>
</tr>
<tr>
<td>Insufficient data</td>
<td>3</td>
<td>2</td>
<td>Organize for our client to collect more data. Allow enough time for data collection.</td>
</tr>
<tr>
<td>Difficulty in processing final report</td>
<td>3</td>
<td>5</td>
<td>Use LaTeX instead of MS-Word. Open Office has good reputation for processing long documents.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>We are using Open Office + Mendeley</em></td>
</tr>
</tbody>
</table>
**Non-technical risk**

<table>
<thead>
<tr>
<th></th>
<th>probability</th>
<th>severity</th>
<th>Contingency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problems with supervisor,</td>
<td>1</td>
<td>4</td>
<td>Choose your supervisor wisely. Adopt a more autonomous approach to problem</td>
</tr>
<tr>
<td>lack of communication.</td>
<td></td>
<td></td>
<td>solving.</td>
</tr>
<tr>
<td>Subject appears in publication</td>
<td>1</td>
<td>2</td>
<td>Assess this with supervisor. Low risk as our client prefers confidentiality,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>this may be a trend in industry.</td>
</tr>
<tr>
<td>Loss of productivity /</td>
<td>2</td>
<td>4</td>
<td>Alternate development and writing tasks.</td>
</tr>
<tr>
<td>distraction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Illness</td>
<td>2</td>
<td>2</td>
<td>Include slack in project plan.</td>
</tr>
</tbody>
</table>

**9.6 Further work**

We see four areas of further development in taking this project closer to its industrial application.

**Accurate estimate of CPU computation time.**

The current number of Assembly language instructions and the CPU cycle timings are inaccurate as they are obtained by derivation. To obtain a correct estimate, a classifier including the multi-class voting function should be implemented in C or Ada, compiled to ARM and profiled on a development board.

The fixed point implementation has only been prototyped for the Polynomial kernel. Our estimate assumes that the exponential function function can be implemented with a lookup table and a Taylor series with 22 terms. It remains to be seen that this type of implementation delivers sufficient resolution for the classifier.

**Collect more data in the least represented categories**

The number of vehicles in the Bus / Coach (class 1) and Special Artic. (class 7) is too low to produce a significant training set. Although care has been taken to avoid bias in the results with cross-validation, it would be beneficial to collect more data samples. This will also give a more realistic estimate of the total number of support vectors required for a final implementation.
Refinement of Fixed-point Implementation

Our work on fixed-point implementation is incomplete. There is no evaluation of the precision required in the approximation of the exponential function, and no implementation of the multi-class voting function.

Outlier detection and class boundaries

We discovered that the SMOTE algorithm indirectly warn us of outliers. It may be advisable to develop an outlier detection function using k-Nearest Neighbours. The purpose of this is to assist human classification of the training sets.

It is probably worth defining a separate class for Bus / Coach + Trailer.

We also noted that certain class boundaries are rather arbitrary, such as pickup trucks with trailers currently classified as heavy-van with trailer (class 6). Their waveform is similar to a car or light-van with trailer (class 4). This is does not directly imply a loss of performance with the Gaussian Kernel, but is worth investigating.
10 References and Bibliography

10.1.1 Textbooks


10.1.2 Machine Learning


10.1.3 Hardware and Software


11 Appendix A: Content of CD-ROM

11.1 Source code and Unit tests

The CD-ROM contains a complete set of Python libraries required for the execution of the source code. The libraries in the /lib-win32 folder can be installed on a Windows-7 32bits operating system. Linux users who are using a Debian or Ubuntu distribution are advised to install the numpy, scipy, and scikit-learn libraries from neuro.debian.net

The /source folder contains all the python modules organised as follows:

<table>
<thead>
<tr>
<th>module</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch_boost.py</td>
<td>Batch processor for the AdaBoost classifier.</td>
</tr>
<tr>
<td>batch_svm.py</td>
<td>Batch processor for the SVM classifier. Includes grid search plot for SVM parameters.</td>
</tr>
<tr>
<td>classify.py</td>
<td>Classifier code, including class-imbalance and SMOTE</td>
</tr>
<tr>
<td>fixed_point.py</td>
<td>Prototype fixed-point implementation</td>
</tr>
<tr>
<td>plot.py</td>
<td>Collection of functions to plot waveforms, spectrum, input vectors.</td>
</tr>
<tr>
<td>preproc.py</td>
<td>csv data serializer, signal pre-processing and serialization of input vectors for SVM and AdaBoost classifiers</td>
</tr>
<tr>
<td>test_batch_boost.py</td>
<td>Black-box test for AdaBoost batch-processor</td>
</tr>
<tr>
<td>test_batch_svm.py</td>
<td>Black-box test for SVM batch-processor</td>
</tr>
<tr>
<td>test_preproc.py</td>
<td>Unit test for parsing of Clearview XML metadata file.</td>
</tr>
</tbody>
</table>

The black-box tests contain a collection of examples of classification as they would be executed from the command-line. The matching JSON batch files are in the /source/test/batch folder.
11.2 Batch files and raw results

The `/source` folder also contains all the metadata and test data required to run tests and experiments.

Batch files and results of the experiments ran during the writing of the dissertation have been moved to a separate folder called `/experiments-data`

11.3 Reports of Experiments

Due to the large number of experiments, results have been collected in spreadsheet documents. These were originally produced in Open Office Calc, and converted to MS-Excel for readers who don't have OpenOffice or LibreOffice installed. Please note that some of the format options for the diagrams are lost in the conversion process.

The files are in `/documents/reports-ms_excel` and `/documents/reports-open_office`

We give here the correspondence between the spreadsheets and the sections in the dissertation:

<table>
<thead>
<tr>
<th>Dissertation Section</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2 First Approach to SVM classification</td>
<td>7_2_first_approach_SVM.ods</td>
</tr>
<tr>
<td>7.3 AdaBoost</td>
<td>datasetC_adaboost.ods</td>
</tr>
<tr>
<td>7.4 Class Imbalance</td>
<td>datasetC_poly_ovo_report.ods</td>
</tr>
<tr>
<td></td>
<td>datasetC_poly_ovr_report.ods</td>
</tr>
<tr>
<td></td>
<td>datasetC_rbf_ovr_report.ods</td>
</tr>
<tr>
<td>7.6 Results with New Dataset</td>
<td>datasetD_poly_ovo_report.ods</td>
</tr>
<tr>
<td></td>
<td>datasetD_poly_ovr_report.ods</td>
</tr>
<tr>
<td></td>
<td>datasetD_rbf_ovr.ods</td>
</tr>
<tr>
<td>7.7 Conclusion and Comparisons</td>
<td>best_performers_report.ods</td>
</tr>
<tr>
<td></td>
<td>best_svm_classification_matrix.xlsx</td>
</tr>
</tbody>
</table>
11.4 **ARM Assembly CPU cycle count**

Finally, the estimate of ARM Assembly implementation and the calculation of CPU cycles can be found in `/documents/CPU cycle count`. Alongside with the spreadsheet to optimize the number of bits in the integer and floating point parts of the fixed-point computation,