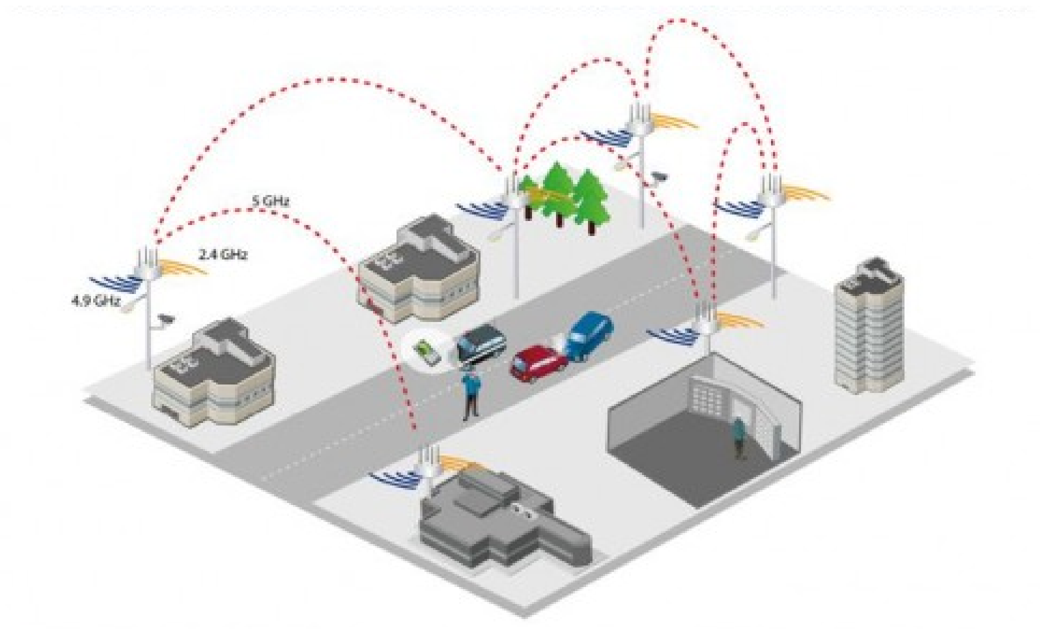
University Of Waterloo

Faculty Of Engineering
Mechanical And Mechatronics Engineering

VIDEO TRANSCODING RATE PREDICTION USING MACHINE LEARNING



Self Study Report

Prepared by
Satya Prahlada Sthita Prajna Kandarpa
UW ID 20402024 | Userid *spspkand*
4B Mechatronics Engineering
4 April 2016

41 Pineslope Crescent
Scarborough, Ontario, Canada
M1E 4M5 *4 April 2016*

Professor William Melek, Director of Mechatronics Engineering
Department of Mechanical and Mechatronics Engineering
University of Waterloo, Waterloo, Ontario
N2L 3G1

Dear Sir,

This report, titled ”Video Transcoding Rate Prediction using machine learning techniques”, was prepared as my 4B Work Report for the University of Waterloo. This report is in fulfillment of the course WKRPT 400.

I got exposed to some innovative and novel media processing techniques at a startup I worked at which helped pique my interest in audio visual media processing. The purpose of this report is to offer a software based predictive model that can decrease the time required to choose video standards for best transcoding performance without actually performing the transcoding operation between an input and output video standard. Such predictive models are useful for extremely large video transcoding operations, usually done by high profile video streaming and storage sites, where its not feasible to test every possible input/output video standard to pick the best one for a specific platform or operation.

The technical analysis conducted by me for this purpose incorporates machine learning techniques to evaluate the standard video encoding technologies to produce models which are able to predict transcoding rates for some of the most common video codec configurations. In conclusion, this report offers a freely available software based predictive model

This report was written entirely by me and has not received any previous academic credit at this or any other institution.

Sincerely,
Satya Prahlada Sthita Prajna Kandarpa
ID 20402024

Table of Contents

List of Tables

List of Figures

Summary

This report first introduces the basic techniques used in video processing to perform tasks such as capture, encoding and decoding, transport and storage. The main contribution made by this report is the availability of a machine learning trained predictive model that enables the prediction of transcoding times for any new sample consisting of a few input and output parameters for a video. The machine learning techniques used fall under the category of regressive analysis of a numerical quantity, the rate of transcoding in [frames per second or fps]. Video transcoding refers to the process of data exchange between heterogeneous multimedia networks to reduce the complexity and transmission time by avoiding total decoding and re-encoding of a video bitstream.

A publicly available dataset of the transcoding performance of a few thousand youtube videos was analyzed using the software package, **caret**, made for the language R. The steps taken to perform a standard statistical regression analysis of this dataset involved a few steps. This dataset includes characteristics such as input codec, frame-rate, size and output codec, frame-rate, bitrate and most importantly, memory and cpu time taken to transcode the input video to output video. The non-linear regression models trained using the dataset include k-Nearest Neighbors (kNN), Neural Networks (NN), Model Averaged Neural Networks (avNNet) and Multivariate Adaptive Regression Splines (MARS). All of the analyses are implemented using the statistical programming language, R using the packages *caret* and *AppliedPredictive-Modelling.* The results from all the models are evaluated using two metrics, namely R^2 and Root Mean Squared Error, which are the statistical performance metrics for regression models.

In conclusion, this report provides a trained neural network model for interested parties to use and test on their own datasets to be able to gain predictions tailored to the hardware being used to implement their low power video sensor networks. It also recommends that the model be re-trained using a bigger

portion of the youtube dataset. This wasn't done on the first try because of the lack of computational power required to train huge number of samples.

1 Introduction

1.1 Background

Digital videos comprise of most of the Internet traffic carried by networks all across the globe today. The problem of storing and transporting digital video has been a hot topic in research for many years. Currently there are over a hundred different video standards which follow different compression techniques and bit stream patterns. One of the biggest video encoding/decoding open source project, Ffmpeg [6], has eighty-eight video standards listed ,but many new ones are made and developed yearly. Also some standards are proprietary and do not have open standards, for example: VP6. The current problem is mobile devices have limited amount of resources to support all the video standards so usually only one is implemented [7].

A few entities like content delivery networks(CDN) and video ad delivery networks are interested in finding the best performing video parameters for optimal transcoding performance. When one quantifies the input and output parameters of this problem as either numerical or categorical variables, it can be thought of as a complex non-linear system of equations that maybe solved to find the most optimal solutions.

Recent advances in machine learning (ML) and pattern recognition has made this class of techniques a flexible solution to simulate natural/artificial dynamic systems without using complicated models that seek to emulate the actual physical processes. The utility of machine learning lies in its ability to pick up on naturally occurring relationships in data (of multiple sorts). Although, in reality, it does come with its own set of pitfalls such as the curse of dimensionality, over-fitting and others. Each of the widely available machine learning techniques are able to model these relationships using smaller functions called basis functions, of which there are many types with differing behaviors.

1.2 Scope

The advantage offered by ML models comes at a price, the invariably high amounts of training time required, especially for good predictive performance on huge datasets. The time required to measure the duration of a transcoding operation can be found by using a trained predictive model given a few input and output parameters of a video. This method can predict the transcoding duration independently of the magnitudes and categorical values of the input and output parameters, i.e., it is a O(1) operation (constant time) in programming performance parlance.

This report explains the steps taken to train such a predictive model using a publicly available dataset of video characteristics for a transcoding process from a set of codecs to another. Multiple models are used to cross-validate predictive performance and setup the a scaffolding for boosting or bagging of different models to offer even better predictions. This report limits its scope to offer a trained predictive model free of use to anyone intending to predict transcoding rates. The techniques used in this report may also be extended in order to predict transcoding performance for a distributed server configuration. However, such augmentations are deemed to be out of scope for this report in spite of their relevance for large server farms handling video transcoding operations.

2 Video Processing And Transport

2.1 Anatomy of a video

Digital videos are ubiquitous in the era of endless streaming/download/playback services such as Youtube, Netflix and VLC.

Digital video is an ordered sequence of digital images, known as frames, played in succession at a given rate, usually represented as a framerate (frames per second or fps).

$$I_1 = \begin{bmatrix} 0&1&2&2&2&2&3&5&7&7\\ 0&0&1&2&2&3&5&7&7&7\\ 0&0&0&1&2&3&5&7&7&7\\ 0&0&0&2&2&3&6&7&7&7\\ 0&0&0&2&2&3&7&7&7&7\\ 0&0&0&0&1&2&5&6&7&7\\ 0&0&0&0&1&2&3&5&6&7\\ 0&0&0&0&1&2&3&5&6&7\\ 0&0&0&1&1&1&2&4&5&7\\ 0&0&1&1&1&1&2&3&4&6 \end{bmatrix}, I_2 = \begin{bmatrix} 0&1&2&2&2&2&3&5&7&7\\ 0&0&0&2&2&3&5&7&7&7\\ 0&0&0&0&2&3&5&7&7&7\\ 0&0&0&0&2&3&6&7&7&7\\ 0&0&0&0&2&3&7&7&7&7\\ 0&0&0&0&0&1&5&6&7&7\\ 0&0&0&0&0&1&3&5&6&7\\ 0&0&0&0&0&1&3&5&6&7\\ 0&0&0&0&1&1&2&4&5&7\\ 0&0&0&0&1&1&2&3&4&6 \end{bmatrix}$$

Figure 2.1 – Matrix representation of a video and its component frames

A grayscale video, represented by V, is a sequence of images

$$V = I_1, I_2, ...I_n, n = \text{number of frames in the video}$$

, and

$$I_k \mid k = 1...n$$

is the matrix representation of an image of dimension $a \times b$. Please refer to Fig. 2.1 for a visual representation of the matrix images. Each image, I_k consists of grayscale(brightness or intensity) values from a finite set C of size c, where

$$C = \{x \mid x = 0, 1, 2, ...N_c - 1\}$$

. A pixel is the basic unit of processing in images. Its location in a video maybe denoted by the 3-D co-ordinates

$$(k, m, n)\text{where } (k, m, n) = (\text{frame number, row number, column number})$$

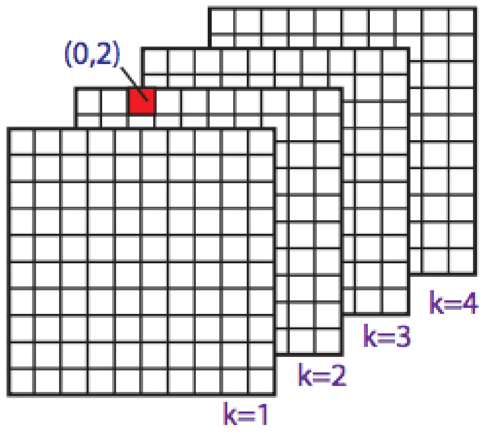


Figure 2.2 – Location of pixel $I_2(0,2)$ in a video frame

The 3-D co-ordinates may also be represented by $I_k(m,n) \in C$. A visual representation of a pixel $I_2(0,2)$ is shown in Fig. 2.2[1].

Videos with color information use a similar representation with an additional color component. Pixels in color video frames may be represented by

$$P(I_k, C_t, m, n)$$

where C_t represents the color component numbered t. For example, an RGB image can have three possible values for t, i.e., $t \in (1,2,3)$. So, $P(I_k, C_t, m, n)$ represents the value of the color component C_t for the pixel with frame co-ordinates (m,n) and frame I_k [8].

Videos can be said to have two main representations in digital media - Pre-recorded videos and live/streamable videos. Disk based videos are playable files that may be stored on a personal computing device or a cloud server. These are binary representations of the video data, obtained by compressing raw video frames to achieve optimal spatiotemporal data representation, i.e., reduce redundant data in frames using a combination of motion tracking, Fourier or Discrete Cosine Transforms, Quantization and Variable Length Encoding [2].

2.1.1 Pre-recorded Videos

Disk based video file formats may contain uncompressed video footage (RAW format) or encoded video footage (MP4, AVI, etc. formats). Most consumer focused video file formats consist of the following components:

Container

The container stores the video and/or audio data using separate encoding formats for video and audio. Popular container types include Matroska(MKV), FLV, Ogg, AVI, etc. It is to be noted that container selection constrains the available video encoding formats. The following table lists a few popular containers and their supported encoder formats.

Name	File Extension	Container	Coding Formats
MPEG-4(MP4)	.mp4	MPEG-4 Part 12	H.264
Matroska	.mkv	Matroska	Any
Flash Video(FLV)	.flv	FLV	H.264, VP6

Table 2.1 – Some common video containers and compatible video coding formats

Video coding(encoding) format

A video coding format (or sometimes video compression format) is a content representation format for storage or transmission of digital video content (such as in a data file or bitstream). Examples of video coding formats include MPEG-2 Part 2, MPEG-4 Part 2, H.264 (MPEG-4 Part 10), HEVC, Theora, Dirac, RealVideo RV40, VP8, and VP9.

2.1.2 Live/Streaming Videos

Streamable videos are defined as multimedia that is constantly received by and presented to an end user while being delivered by the provider. Streaming refers tot he delivery method of the video, rather than the video itself, and is an alternative to downloading a full video file. This report deals specifically with live streaming videos, which involves a source media type, a screen recorder in this case, an encoder to digitize the content, and a transport medium, usually one of HTTP, RTSP or RTP.

The main difference between downloadable and streamable videos is speed with which the end user may start watching the video. In case of downloadable videos (files), the user has to wait till the entire file has downloaded to be able to start playing the video. Streamable videos, however, make use of video codecs that are tailored to give the option of beginning playback from any position. They can make this happen by using multiple frame types, frame prediction methods. One frame type in particular, known as a keyframe, enables this resume capability of video from any position because of its decoupled nature from preceding and succeeding frames. Keyframes are implemented differently by video codecs but their essential function stays the same across all codecs. H.264, has a structure that enables interoperability during the video decode process with older video standards. The operational specifics of H.264 are explained in detail in Section 2.2.2.

2.2 Video Processing

Video processing consists of three main processes - Encoding, Decoding and Transcoding

2.2.1 Encoding and Decoding

Encoding involves the analysis of uncompressed video files (RAW format) to remove redundant and/or visually indiscernible data and generate a bitstream representation of the video. This bitstream representation may then be used to generate files and/or streamable videos. Decoding involves recovering a playable (streamable) video from the bitstream generated by the encoding process. A *video codec* is a program that can perform both encoding and decoding of a video or bitstream respectively.

The general structure of a video codec is shown in Fig. 2.3. It is important to note that the network transport stage of a streamable video involves sending this bitstream representation of a video to an end-user's browser or video playback application like VLC or Quicktime. Decoding occurs in the end-user application via available software or hardware codecs.

This process of data reduction is called video compression or encoding.

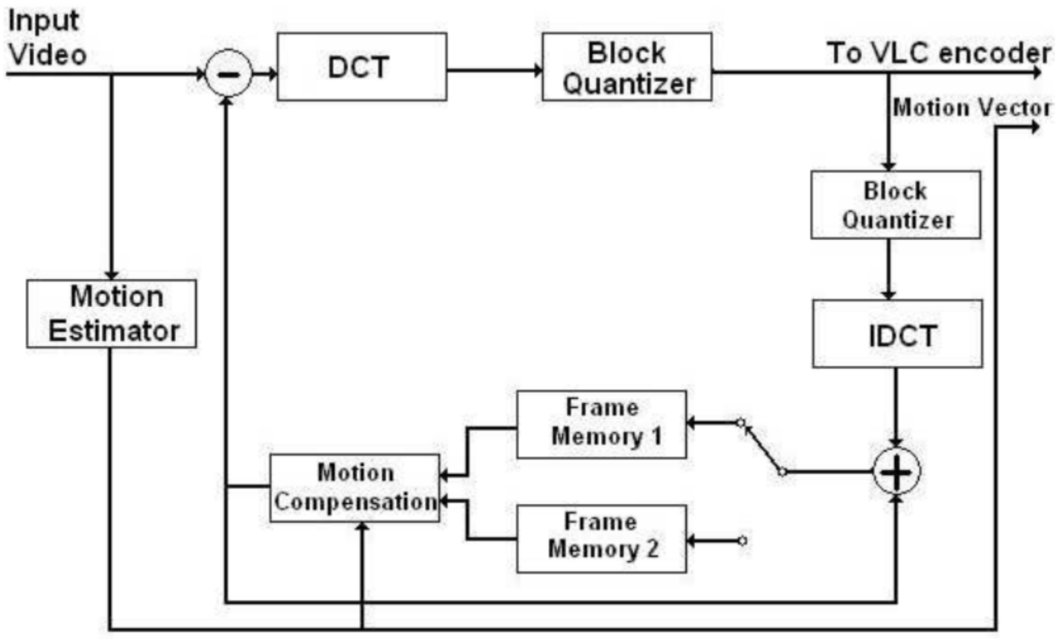


Figure 2.3 – Structure of a video codec

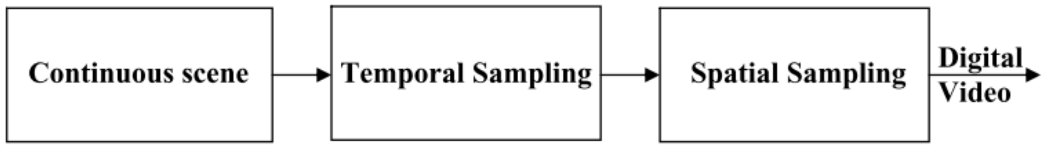


Figure 2.4 – Digital video produced through compression techniques

When videos are captured by a camera, they are usually stored in an uncompressed format where each frame contains all the original data recorded by the capture device. This process is shown in Fig. 2.4. As evident in the figure, there are two sampling subprocesses, namely Spatial Sampling and Temporal Sampling, that are executed serially to produce a compressed video.

The data present in video frames can have 4 kinds of redundancies [9].

Temporal Redundancy

Temporal Redundancy: Since the elapsed time between two consecutive frames is generally very short, consecutive frames tend to be very similar in content and thus, contain data redundancy. The differences between consecutive frames may be expressed by considering the displacements of objects in the frames and encoding this motion′s vectors and differences.

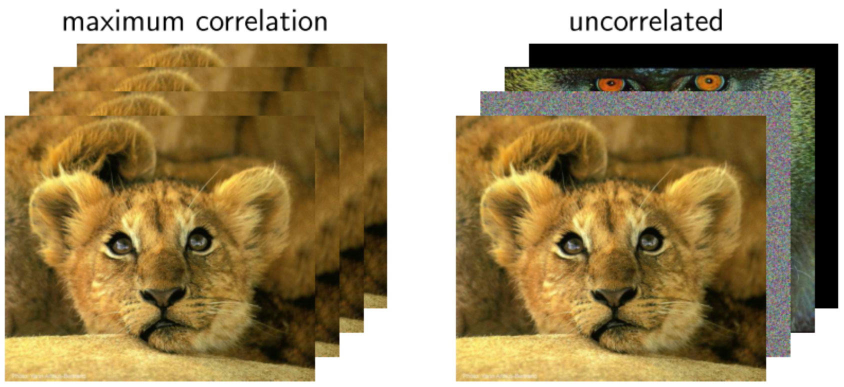


Figure 2.5 – Demonstration of temporal redundancy[1]

To simplify this procedure, a frame is divided into small fixed (H-261, MPEG-1 encoders) or variable sized blocks(H.263, MPEG-4, H.264 encoders). Motion detection may then be performed by using a statistical measure to determine the best match for a block in a window centered at the block′s position in the second frame [2].

Psycho Visual Redundancy

Psycho Visual Redundancy: Since the target audience for 99.99% of all videos is a human recipient, the capabilities of the human visual system (HVS) need to be taken into account before encoding. The HVS is very sensitive to changes in luminance aka intensity compared to changed in chromaticity (color). In fact, the HVS is extremely good at inderring color details based on the intensity levels in an image. This knowledge maybe used to selectively subsample the color data in a frame while keeping the intensity data unchanged.

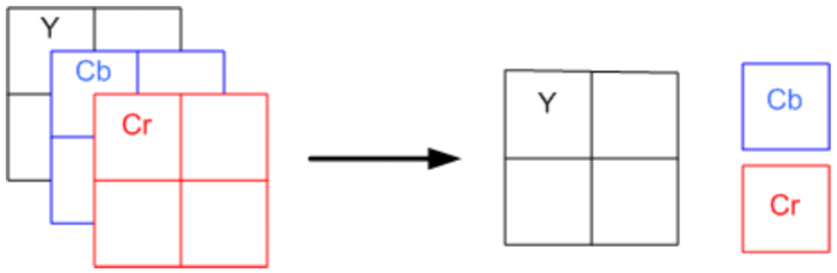


Figure 2.6 – Subsampling chromaticity to achieve data reducton[2]

A frame maybe divided into macro blocks which are further divided into three layers with each layer holding one of the color components of the macroblock pixels. YCbCr is the color space used in almost all video coding standards because of its compatibility with the YUV color space used in most displays and televisions and its ability to separate chromaticity from intensity. If the macro block layer for each of Y, Cb and Cr components has 4 8x8 blocks, the Cb and Cr components can each be subsampled to 1 8x8 blocks. The format obtained in this way is referred to as 4:2:0 and the subsampling creates a data reduction of about 50%. A macro block converted to 6 blocks of 8x8 each using 4:2:0 mode is shown in Fig. 2.6

Spatial Redundancy

In any given frame, a pixel′s value is correlated to its neighboring pixel values most of the time. Thus, this value maybe predicted to a certain extent given the values of its neighboring pixels. An example is shown in Fig. 2.7. High correlation means that pixels within a neighborhood have similar colors and zero correlation can mean that pixels in a neighborhood are unrelated in color. Spatial redundancy may be reduced by using transforms such as Discrete Co-

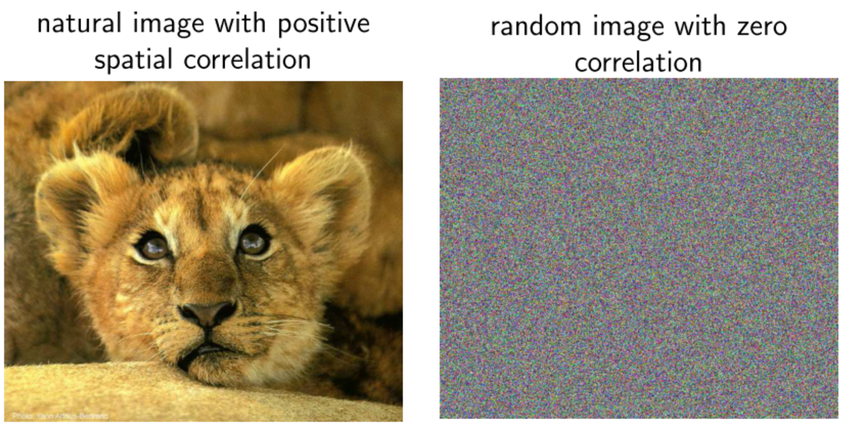

natural image with positive spatial correlation
random image with zero correlation

Figure 2.7 – Demonstration of spatial redundancy[1]

sine Transform (DCT) or Discrete Wavelet Transform (DWT). These values are then quantized and converted to 1-D vectors by reading their values in zig-zag order. These transforms eliminate high frequency pixel values with low energy content.

The quality of the image/frame is directly related to this elimination, which means a trade-off between quality and compression ratio can be achieved based on constraints imposed by the transmission/playback medium of the video. A video meant to be consumed from disk based file systems can be allowed to retain more data during encoding while videos created for a streaming medium would need to take network bandwidth into consideration to determine an optimal compression ratio.

In case of our application, the video is meant to recorded, encoded and sent over the network like a streamable video. So, higher compression ratios are desired while still being able to maintain video resolution and clarity rivaling HDMI (720p HD or 1280x720 frame dimensions).

The process followed to achieve an optimum compression ratio is explained in later sections.

Statistical Redundancy

The process of reducing statistical redundancy is known as entropy coding. Entropy coding needs to occur after spatial redundancy reduction for optimum compression. The quantized frame data obtained after spatial redundancy reduction is then compressed by Run Length Encoding (RLE) and the resulting values are coded (each unique value gets a unique binary representation) using Huffman encoding.

Live video considerations

Since temporal redundancy reduction makes use of the differences between consecutive frames, this may result in an accumulation of errors even if a frame experiences corruption during network transport.

This scenario can happen if the network transport method used is UDP (User Datagram Protocol), as UDP does not guarantee packet delivery and correct order of packet delivery as is the case with the TCP/IP protocol (used by HTTP). However, UDP is extremely useful for low latency data transmission due to its lack of error correction mechanisms that can guarantee packet delivery without corruption. The operational nature of UDP is commonly referred to as ”send and forget”.

Encoding methods that depend only on the previous frame create a serially accessible frame sequence that requires that the end user download and decode all frames before a particular frame to be able to correctly view the frame. This drawback has been overcome in various ways by different codecs. For example, the MPEG-2 codec uses multiple types of frames, with each type differentiated by the amount of data they hold and dependence on previous or future frames.

Further improvements to video codecs resulted in the development of the H.264/AVC standard described in the following section. The performance and interoperability offered by the H/264 encoder absolutely blew every other encoder out of the water, especially for applications dependent on network transport.

2.2.2 H.264/AVC encoder

The H.264 encoder includes a set of improvements to the video coding process that provides enhanced compression performance relative to other encoders like MPEG-2 and VP6. The enhancements provided by H.264 specifically target broadcast, streaming, video telephony and other network friendly video representations. It provides significant improvement in rate distortion efficiency relative to existing standards [10]. All of these features have enabled the H.264 codec to sort of become the de facto standard for video compression for network streaming applications.

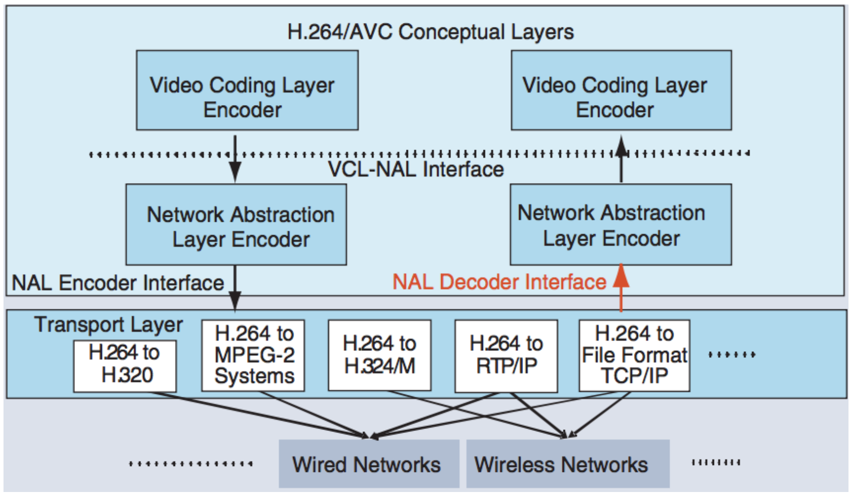


Figure 2.8 – Structure of H.264/AVC encoder[3]

A general architecture of the H.264/AVC codec is provided in Fig. 2.8. For efficient transmission in different environments not only coding efficiency is relevant, but also the seamless and easy integration of the coded video into all current and future protocol and network architectures. This includes the public Internet with best effort delivery, as well as wireless networks expected to be a major application for the new video coding standard. The adaptation of the coded video representation or bitstream to different transport networks was typically defined in the systems specification in previous MPEG standards or separate standards like H.320 or H.324. However, only the close integration of network adaptation and video coding can bring the best possible performance of a video communication system.

Therefore H.264/AVC consists of two conceptual layers. The video coding layer (VCL) defines the efficient representation of the video, and the network adaptation layer (NAL) converts the VCL representation into a suitable format for specific transport layers or storage media. For circuit-switched transport like H.320, H.324M or MPEG-2, the NAL delivers the coded video as an ordered stream of bytes containing start codes such that these transport layers and the decoder can robustly and simply identify the structure of the bitstream. For packet switched networks like RTP/IP or TCP/IP, the NAL delivers the coded video in packets without these start codes [3].

The following features describe the important features of the H.264 codec that make it a better choice over previous coding standards:

Intra Prediction

Intra prediction means that the samples of a macroblock in a frame (slice, in case of H.264) are predicted by using information of already transmitted macroblocks of the same frame. It is to be noted that each image is divided up into smaller packets (NALs) which can be read into macroblocks. H.264 uses varying modes for Intra Frame Prediction depending upon the rates of change of luminance and chromaticity in the image.

Motion Compensated Prediction

This is a form of inter-frame (image) prediction. In this case, the macroblocks of an image can be predicted from already transmitted macroblocks of previous reference images. H.264 differs from previous standards (specifically, MPEG) in that it can use several preceding reference images for motion compensation prediction. For this purpose, an additional picture reference parameter has to be transmitted along with the standard motion displacement vectors usually needed for motion compensation prediction as described in Section 2.2.1.1.

Block Transform Coding

Former standards such as MPEG-1 and MPEG-2 used a Discrete Cosine Transform (DCT) with block size 8x8 for the purpose of transform coding. H.264 mainly uses 4x4 block sizes while switching to 2x2 blocks in special cases. It also uses 3 different kinds of applied integer transforms instead of a DCT. The first transform type of size 4x4 is applied to all samples of luminance and chromaticity components regardless of whether motion compensation prediction or intra prediction was applied. The other two types of transforms are Haddard transforms of sizes 4x4 and 2x2 respectively.

Compared to the DCT, the applied integer transforms used in H.264 have only integers between -2 and 2 in their transform matrix. This allows computing the transform and inverse transform in 16-bit arithmetic using only low complexity shift, add and subtract operations [3].

Entropy Coding Schemes

Entropy coding is used to reduce statistical redundancy, i.e., use lower number of bits to represet values that occur with high frequencies and a high number of bits to represent values that occur with low frequencies. This reduces the amount of data needed to represent the overall data required to make up the data.

2.2.3 Transcoding

Video transcoding refers to the process of data exchange between heterogeneous multimedia networks to reduce the complexity and transmission time by avoiding total decoding and re-encoding of a video bitstream. Despite the fact that a video stream is generated by eliminating all redundancies, many network channels may not have the necessary capabilities to handle these streams. This restriction may be overcome by reducing the video data size through a change in video format. In terms of video properties, this change can be affected by changing bits per pixel, pixels per frame (pixel density reduction), frames per second, video content or coding standard [2].

Video transcoding for real-time applications on raw video data is extremely time consuming because of the motion estimation and data transformation operations. Acceptable transcoding performance for real time operations can be achieved however, if the conversion of video formats is performed on compressed data rather than raw data. A few effective compressed data video transcoding techniques include:

- Bitrate transcoding
- Spatial transcoding
- Temporal transcoding
- Standard transcoding

A description of these techniques was deemed to be beyond the scope of this report, but more information may be found in the paper cited here [2].

In summary, it can be said that transcoding is something of an art form whereby one must balance dozens of requirements, formats, parameters and more. General video transcoding best practices are presented as follows:

- Always encode for a specific quality rather than relying on bitrates. With bandwidth availability increasing across the board there is no need for using a target bitrate unless a specific limited device is being targeted or the quality required is unrealistic within bitrate constraints (in which case quality expectations have to be lowered)

- Avoid upscaling video dimensions from the original dimensions as this only blurs the video. In general, video players do automatic upscaling to make the video fit device screen, so it isn't necessary to do this during transcoding.

- Using Free and Open Source software like FFmpeg and libx264[11] is highly recommended. These libraries have a community of video experts offering help and are very friendly. A lot of money may be saved by not using licensed and proprietary tools while still being able to provide insanely good video quality.[12]

3 Machine Learning based Regression Model Training

3.1 Dataset Characteristics

The dataset found at [13] can be used to gain insight into characteristics of consumer videos found on UGC(Youtube). The features include bitrate, framerate, resolution, codec, number of i frames, number of p frames, number of b frames, size of i frames, size of p frames and size of b frames of the input video and the desired bitrate,framerate, resolution and codec of the output video which are given as a parameter to a transcoding service.

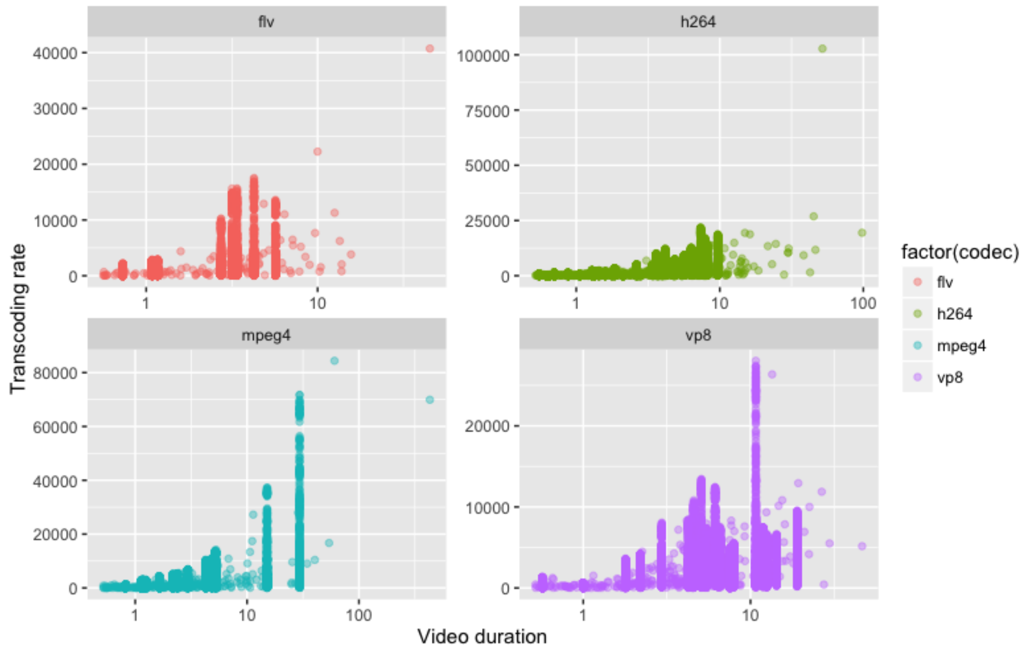


Figure 3.1 – Transcoding rate [fps] v/s video duration [min]

The second file of the dataset contains 20 columns(see column names for names) which include input and output video characteristics along with their transcoding time and memory resource requirements while transcoding videos to different but valid formats. The second dataset was collected based on experiments on an Intel i7-3720QM CPU through randomly picking two rows from the first dataset and using these as input and output parameters of a video transcoding application, Ffmpeg.

Please refer to Appendix A.2 for a thorough visualization based characterization of this dataset. A through exploration of this dataset, including scatterplots using the R package *AppliedPredictiveModelling* [14], threw up no single parameter as being primarily responsible for dictating the transcoding time of a video. This pointed towards the need for a machine learning based approach that can run through multiple regression models while tuning parameters in fine and small amounts to provide the best fit possible. The sheer number of input parameters to tune and account for to be able to predict one final output variable, the transcoding time, also led to the choice of using machine learning algorithms for prediction purposes.

3.2 Methodology for Regression based training

The use of complex classification and regression models is becoming more and more commonplace in science, finance and a myriad of other domains. Regression is concerned with modeling the relationship between variables that is iteratively refined using a measure of error in the predictions made by the model. Regression methods are a workhorse of statistics and have been cooped into statistical machine learning. This may be confusing because one can use regression to refer to the class of problem and the class of algorithm. Really, regression is a process.

Mathematically, we can define the process of Regression modeling as follows. Suppose, we have an output Y and a series of inputs or predictors (usually assumed to be independent variables).

$$X_1, X_2, ...X_n$$

Then, the goals of a regression model are multiple:

1. examine the relationship between inputs and outputs – Do they tend to vary together? What does the structure of the relationship look like? Which inputs are important?

2. Given a new set of predictor values $X_1^*, ...X_p^*$, what can be said about an unseen Y^*?

3. Regression tools often serve as a building block for more advanced methodologies - Smoothing by local polynomials, for example, involves fitting lots of regression models ”locally”, while iteratively fitting weighted regressions is at the heart of the standard computations for generalized linear models

Machine learning is a subfield of computer science that evolved from the study of pattern recognition and computational learning theory in artificial intelligence. Machine learning explores the study and construction of algorithms that can learn from and make predictions on data. Such algorithms operate by building a model from example inputs in order to make data-driven predictions or decisions, rather than following strictly static program instructions.

Regression analysis is widely used for prediction and forecasting, where its use has substantial overlap with the field of machine learning. Regression analysis is also used to understand which among the independent variables are related to the dependent variable, and to explore the forms of these relationships. In restricted circumstances, regression analysis can be used to infer causal relationships between the independent and dependent variables. However this can lead to illusions or false relationships, so caution is advisable; for example, correlation does not imply causation. In a narrower sense, regression may refer specifically to the estimation of continuous response variables, as opposed to the discrete response variables used in classification. The case of a continuous output variable may be more specifically referred to as metric regression to distinguish it from related problems.

3.3 Evaluation of Software Packages

Two software library ecosystems were evaluated for the implementation of the regression based model training. It can be said that the ecosystem of packages and maturity is pretty good across all the platforms due to the explosion in popularity of machine learning techniques across all domains.

3.3.1 Python - *scikit-learn*

The scikit-learn project provides an open source machine learning library for the Python programming language. The ambition of the project is to provide efficient and well-established machine learning tools within a programming environment that is accessible to non-machine learning experts and reusable in various scientific areas. The project is not a novel domain-specific language, but a library that provides machine learning idioms to a general purpose high-level language. Among other things, it includes classical learning algorithms, model evaluation and selection tools, as well as preprocessing procedures.

All objects within scikit-learn share a uniform common basic API consisting of three complementary interfaces: an estimator interface for building and fitting models, a predictor interface for making predictions and a transformer interface for converting data [15].

3.3.2 R - *caret*

The caret package, short for classification and regression training, was built with several goals in mind:

1. to eliminate syntactical differences between many of the functions for building and predicting models,
2. to develop a set of semi-automated, reasonable approaches for optimizing the values of the tuning parameters for many of these models and
3. create a package that can easily be extended to parallel processing systems.

The package contains functionality useful in the beginning stages of a project (e.g., data splitting and pre-processing), as well as unsupervised feature selection routines and methods to tune models using resampling that helps diagnose over-fitting. *caret* depends on over 25 other packages, although many of these are listed as ”suggested” packages which are not automatically loaded when caret is started. Packages are loaded individually when a model is trained or predicted.

The author of this report had invested a lot of time in the past year in learning and getting familiar with the R ecosystem, through his work at another startup that dealt primarily with data analytics for realtime embedded communication systems, Acerta Analytics. Thus, it was decided to go ahead and use R, the *caret [16]* package for regression modelling, *lattice [17]* and *ggplot2 [18]* for visualizations. This decision helped save time and enabled the entire machine learning portion of this report to be complete in 4 days.

3.4 Regression Training using *caret*

Caret has built in methods and and objects to deal with a data set from the first step of analysis, ie., exploratory data analysis to analyze, determine and mark the predictors and the output variables that need prediction. For example, the *featurePlot* function may be used to plot each of the predictor variables against each other and mainly, the output variable. This can help one understand and determine right away, the presence of any linear correlations between the predictor variables and output variable. The plots that the *featurePlot* function can handle include scatterplots, overlayed density plots, box plots that show the extent of linear correlation and scatter plots with overlayed regression line smoothers. The official documentation available at [19] explains and shows a few example plots that help one understand the dataset.

In case of the youtube dataset, the number of predictor variables was found to be 18 and the output variables measured were the memory used for transcoding, *umem* and the CPU time taken for the transcoding process, *utime*. A summary of the elements present inside a data frame (standard R object) is presented below:

3.4.1 Data pre-processing and Splitting

As is evident in the summary presented above, a couple of new columns were added to convert the categorical variables for input and output codecs to integer factors. This had to be done since a few of the models being trained using the dataset are sensitive to the presence of categorical variables and are not really meant to be used with such data frame columns. An example of this can be seen in implementations of the k-Nearest Neighbors algorithm. This may be overcome by using Hamming distance instead of Euclidean distance for calculating distance between neighbors in the algorithm [20].

An other addition to the dataset was the creation of a *trans_ rate* column, which stands for transcoding rate and was calculated with the following formula:

$$transcodingrate = \frac{Num.of frames}{TranscodingCPUtime}$$

This has been done to ensure that the output variable being modeled for is independent of the video size, and thus, can be closer to a good measure of the case where transcoding happens in real-time, aka streaming.

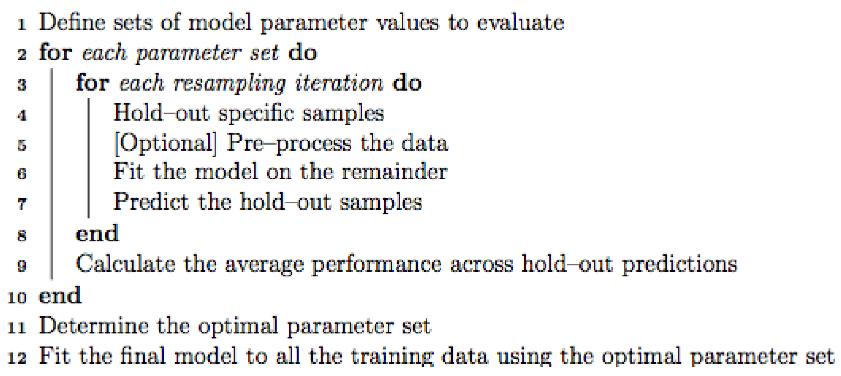
However, the most efficient way of determining the process of sample utilization, according to statistics, requires the usage of all the samples for training a model. Usually, this would result in a model over-fitting for the sample data. However, there are a few techniques that may be used to ensure over-fitting or under-fitting of data does not occur. These techniques, a few of which are cross-validation, bootstrapping, n-fold cross-validation and re-sampling [21].

In the field of statistical data analysis, one of the first tasks is to determine how much of the finite dataset is to be used for model training while ensuring a certain portion of the dataset is kept aside for testing the efficacy of the model after training. Thus, it is important to ensure that a model being trained never gets exposed to the split testing /validation dataset. This is a very good measure of determining how well the model would perform when being used in real life. The main function of the test split dataset is to compare and evaluate performance across models, as most statistical models are actually combinations of localized models.

The *caret* package, specifically has a *createDataPartition* function, that analyses a dataset's characteristics such as multivariate correlation and determines the most randomized way to split the dataset. For regression, the function determines the quartiles of the data set and samples within those groups. The youtube dataset, which consists of about 69000 rows of data, would cause most models to take a lot of time to train with the computing resources available to this student. Thus, 5% of the dataset was randomly sampled to be used as the training data. The rest of the 95% of the dataset was then used to evaluate the performance of the models. A convenience function provided by *caret*, *nearZeroVar* was then used to determine a few non useful predictor variables and these were excluded from the predictors for testing and training datasets. These so-called near zero-variance predictors can cause problems during resampling for some models such as linear regression [21].

This concludes the data pre-processing portion of this analysis. The function used by *caret* to train using a model, the *train* function, takes a *preProc* argument that can be used to center and scale predictor variables, as required by models such as neural networks.

3.4.2 Tuning and building models



```
Define sets of model parameter values to evaluate
for each parameter set do
    for each resampling iteration do
        Hold–out specific samples
        [Optional] Pre–process the data
        Fit the model on the remainder
        Predict the hold–out samples
    end
    Calculate the average performance across hold–out predictions
end
Determine the optimal parameter set
Fit the final model to all the training data using the optimal parameter set
```

Figure 3.2 – Standard operating procedure to tune a model's parameters [4]

The general process that needs to be followed to do efficient parameter tuning while building a model is shown in Figure. 3.2. The *caret* package has several functions that streamline the process of model building, tuning and evaluation. The *train* function can be used to

- Evaluate, using resampling the effects of model parameter tuning on performance
- Choose the optimal model across these parameters
- Estimate model performance from a training set[4]

The models trained the youtube dataset were the following:

k-Nearest Neighbors

k-Nearest Neighbors (kNN) is a non parametric lazy learning algorithm which defers the decision to generalize beyond the training samples till a new query is encountered. For regression, kNN may be used to estimate continuous variables. It works by using a weighted average of the k-nearest neighbors ,weighted by the inverse of their Euclidean or Manhattan distance.

- Parameters and Evaluation metrics:

  The parameters to tune in the kNN model is the number of nearest neighbors used to compute the distance from existing samples for a new sample, **k**.

  kNN maybe evaluated using the Root Mean Squared Error, RMSE.

Neural Networks

Artificial Neural networks are computing systems made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs. Neural networks are typically organized in layers. Layers are made up of a number of interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. The hidden layers then link to an 'output layer' where the answer is output as shown in Figure 3.3.

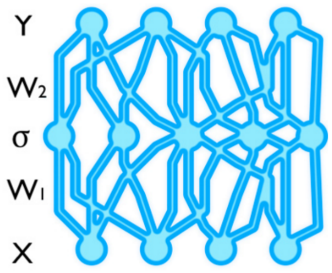


Figure 3.3 – A neural network (a deep learning tool) [5]

- Parameters and Evaluation metrics: An ANN maybe described by two parameters, size and decay. Its performance can be evaluated using regression plots, and RMSE and the coefficient of determination, R^2, which is a measure of how well the model was able to predict each sample.

Multiple Adaptive Regressive Splines (MARS)

MARS is a form of regression analysis introduced by Jerome H. Friedman in 1991. It is a non-parametric regression technique and can be seen as an extension of linear models that automatically models nonlinearities and interactions between variables. The MARS model is a weighted sum of Basis functions,

$$f(x) = \sum_{1=1}^{k} c_i B_i(x)$$

[22]. c_i is a constant coefficient. Each basis function can take three different forms - a constant, a hinge function and a product of two or more hinge functions.

MARS models may also be evaluated using the standard regression measures RMSE and the coefficient of determination, R^2

3.4.3 Resampling and Model Cross-Validation

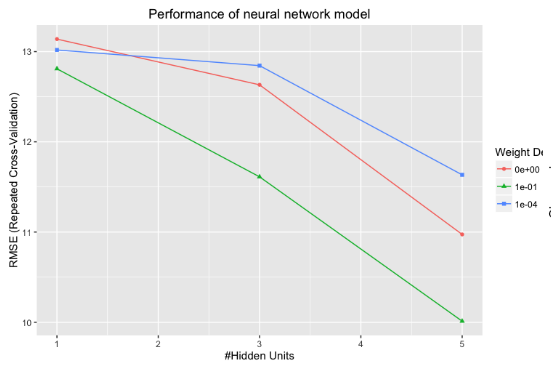
Resampling is the process of creation of modified datasets from the training dataset. Each data set has a corresponding set of hold-out samples. For each candidate tuning parameter combination, a model is fit to each resampled data set and is used to predict the corresponding held out samples. The resampling performance is estimated by aggregating the results of each hold-out sample set. These performance estimates are used to evaluate which combination(s) of the tuning parameters are appropriate. Once the final tuning values are assigned, the final model is refit using the entire training set [21].

For the *train* function, the possible resampling methods are: bootstrapping, k-fold crossvalidation, leave-one-out cross-validation, and leave-group-out cross-validation (i.e., repeated splits without replacement). By default, 25 iterations of the bootstrap are used as the resampling scheme. ALll the models used for the youtube dataset training, used 10-fold cross-validation repeated 10 times to ensure proper fitting of sample data.

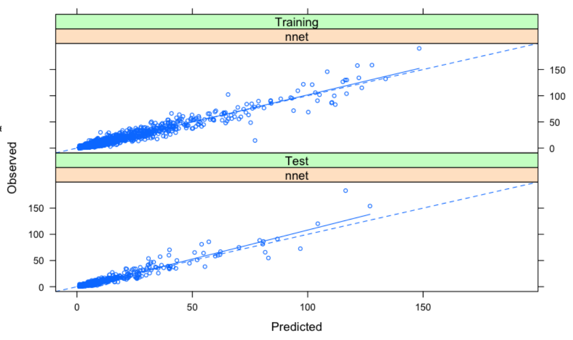
The train function can also perform cross validation, this setting maybe configured in the tuneGrid and trainControl function arguments. Please refer to Appendix A.1 for the actual code that shows these parameters.

3.5 Testing/Training Results

3.5.1 Neural Network Models



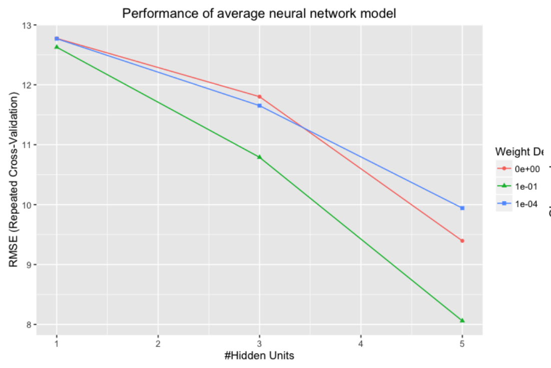
(a) Peformance Tuning



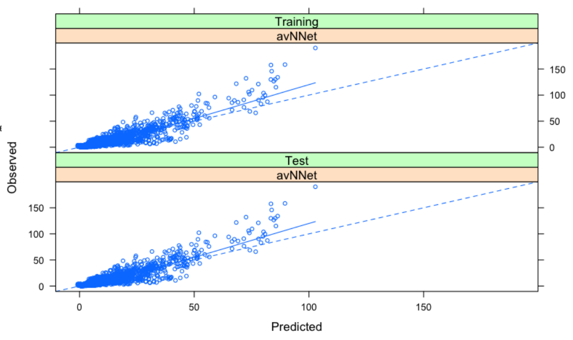
(b) Regression plot

Figure 3.4 – Neural Network Performance Metrics

The neural network model performed exceptionally well in fitting both the training and test datasets to itself. This is evident in Fig. 3.4b. Fig. 3.4a shows how the RMSE changes as the number of hidden units in the neural network are increased. The best performance seems to have happened when using a decoy of 0 and hidden unit size of 5. Please refer to Table 3.1 and Table 3.2 for specific values of performance measures RMSE and R_2.



(a)

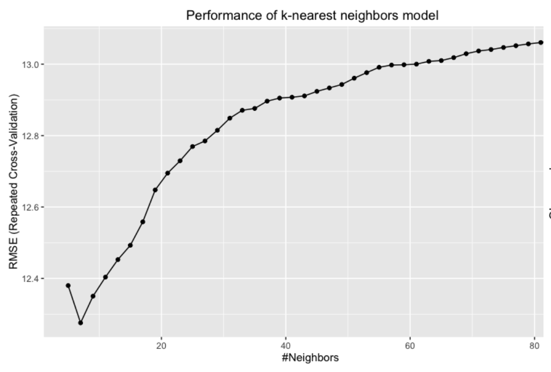


(b)

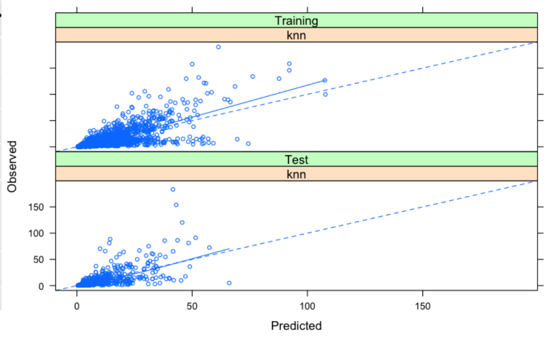
Figure 3.5 – avg Neural Network Performance Measures

The averaged neural network performed slightly worse than a simple neural network on both the training and test datasets. THis may be noticeable because of the amount of variation from the regression line in Fig. 3.5b. Its best performance, when evaluated using the RMSE, seems to have happened with a weight decay parameter of 0.1. Please refer to Table 3.1 and Table 3.2 for specific values of performance measures RMSE and R^2

3.5.2 Nearest Neighbor Models



(a)



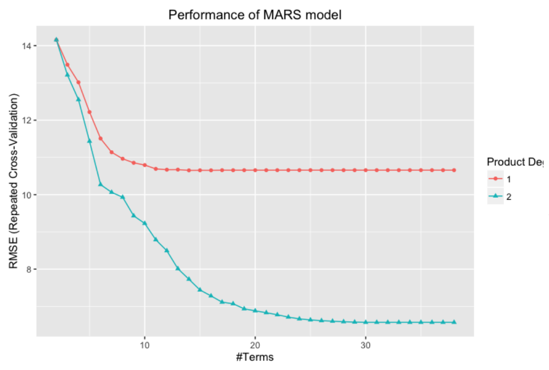
(b)

Figure 3.6 – kNN Performance Measures

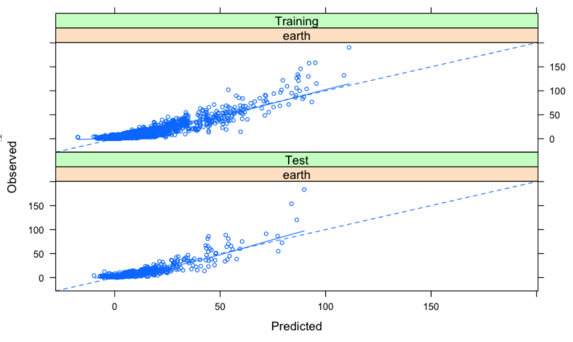
The performance of the k-Nearest Neighbor model decreased while k was increased till $k = 7$. Then its performance exponentially degraded as k increased further higher. Its regression plot, in Fig. 3.6b, also shows a significant portion of both the training and test samples having higher degree of error from the regression line. Please refer to Table 3.1 and Table 3.2 for specific values of performance measures RMSE and R^2

3.5.3 Multivariate Adaptive Regression Splines

The performance of the MARS model almost rivals that of the neural network model. In fact, there is no way to differentiate between their regression fit without referring to R^2 values. Its performance increased exponentially when the number of basis functions was increased with the best performance deemed to have occurred with about 37 basis functions in the models. Please refer to Table 3.1 and Table 3.2 for specific values of performance measures RMSE and R^2



(a)



(b)

Figure 3.7 – MARS Performance Measures

3.5.4 Resampling Statistics

This section shows the variation of the performance measures Root MEan Squared Error (RMSE) and R^2 across different resamples of the training data. In summary, it maybe said that both the MARS neural network models. THe average neural network model seemed to show a more reliable and consistent RMSE and R^2 values across multiple resamples.

Model	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
knn	9.110	11.040	12.010	12.280	13.500	17.210	0
mars	4.833	5.883	6.360	6.572	7.393	9.972	0
nnet	3.661	6.532	10.680	10.010	13.260	17.210	0
avnnet	4.177	6.298	7.465	8.060	9.353	13.460	0

Table 3.1 – RMSE

3.5.5 Resampling Visualizations

Figures. 3.8 and 3.9 are visual representation of the data presented in Table 3.1. Once again, the MARS model is deemed to be the most consistent model because of its low variation of RMSE and R^2 values across resamples. These plots make it very clear that the MARS model fit the training data consistently.

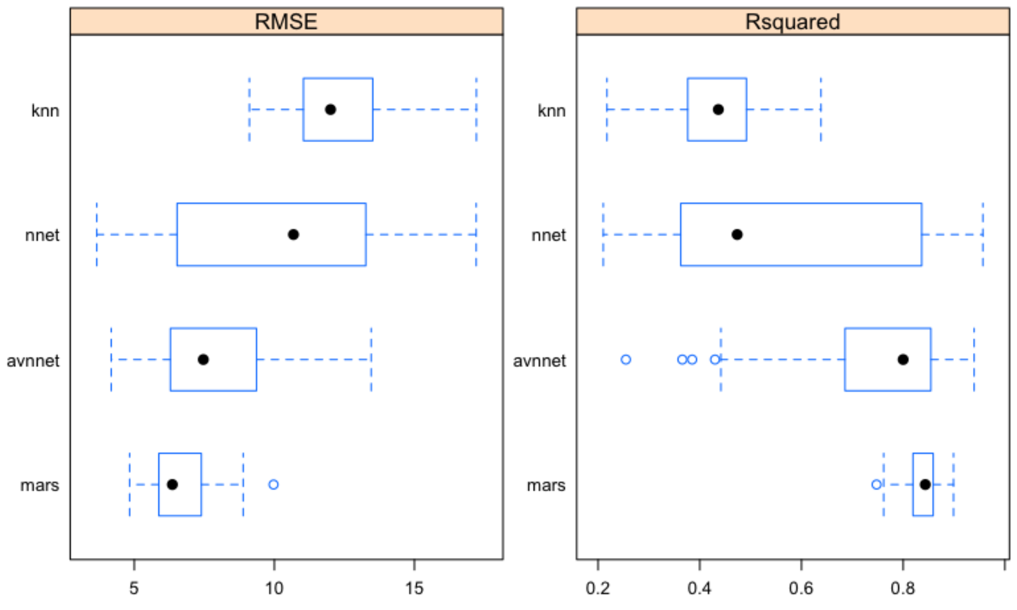


Figure 3.8 – Box whisker plot of RMSE and R^2 variation across resamples

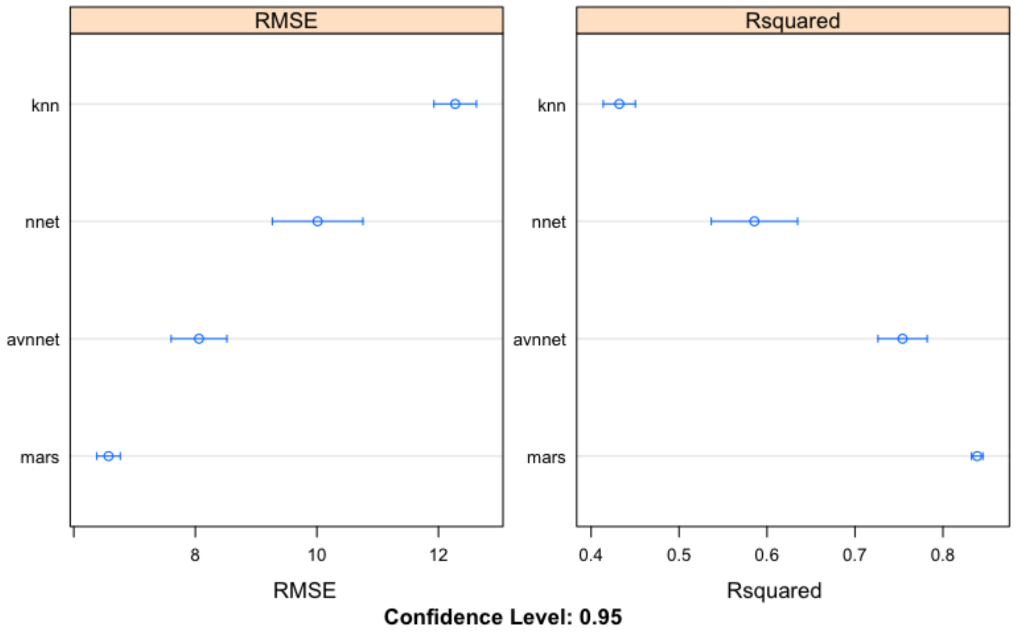


Figure 3.9 – Dot plot of RMSE and R^2 variation across resamples

Figure 3.10 shows how the RMSE varied for each model across resamp runs. Please note that this is a measure of cross-validation across models for each resample of the training data set. This maybe achieved by using *caret's*

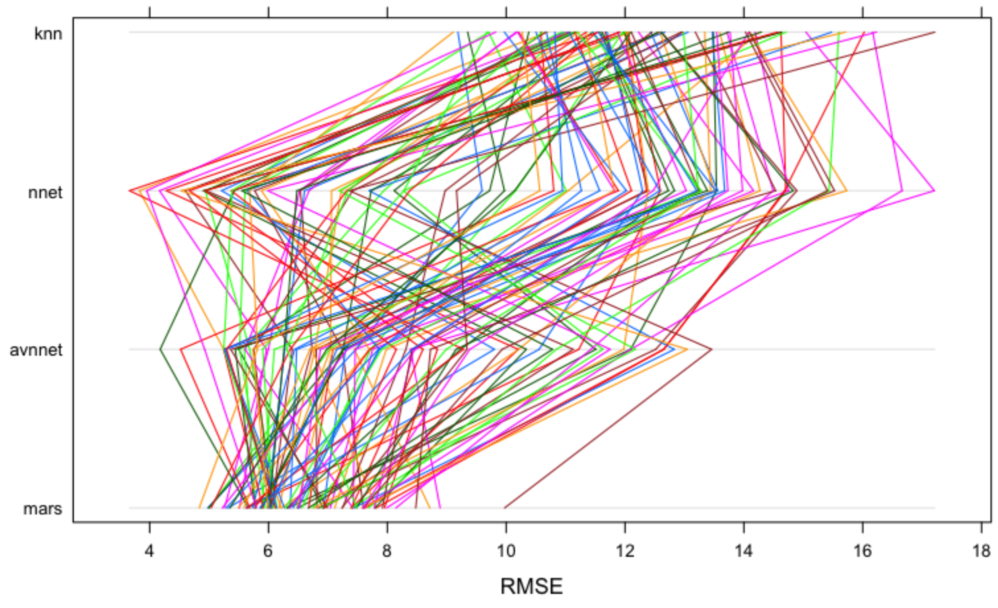


Figure 3.10 – Parallel plot of RMSE and R^2 variation across resamples

resamples function that takes in a list of trained models and a training dataset and a number of iterations of the resample process. All the relevant code for producing this cross-model resampling and visualization is also available in Appendix A.1 towards the end.

3.6 Statistical Performance Measures

3.6.1 Performance evaluation

Table 3.3 shows the best case performance of each model on the training dataset. The neural network comes out on top when evaluated using this data only. MARS comes behind in a close second place.

Table 3.4 shows the best case performance of each model on the test data set. This table is arguably much more important for performance evaluation as it eliminates the possibilities of over fitted models, etc. Once again, the neural network ended with a performance very similar to its performance on the training dataset. COnsidering that the training set was only 5% of the total dataset, this performance is actually extremely good. The MARS model,

Model	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
knn	0.2175	0.3784	0.4366	0.4321	0.4907	0.6384	0
mars	0.7477	0.8197	0.8435	0.8389	0.8585	0.8990	0
nnet	0.2102	0.3626	0.4736	0.5857	0.8351	0.9568	0
avnnet	0.2547	0.6865	0.8000	0.7541	0.8544	0.9396	0

Table 3.2 – R^2

Model	rmse	r2
KNN	10.59699280697047	0.584605561041047
AvgNN	6.95034584039986	0.843654549692612
MARS	6.29845637084742	0.850651878271596
NN	4.01016583316771	0.939547289034280

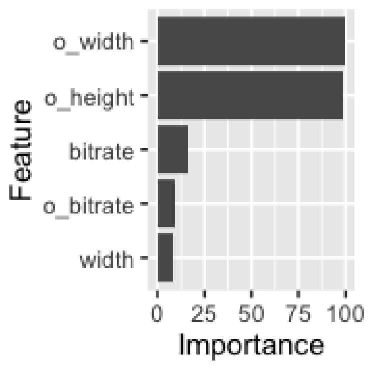
Table 3.3 – Results of model performance on trained dataset

Model	rmse	r2
KNN	12.44409507063652	0.403246960924018
AvgNN	8.21448372643189	0.764592868929468
MARS	6.86267500478760	0.818602643604257
NN	4.81454518621358	0.911059846936685

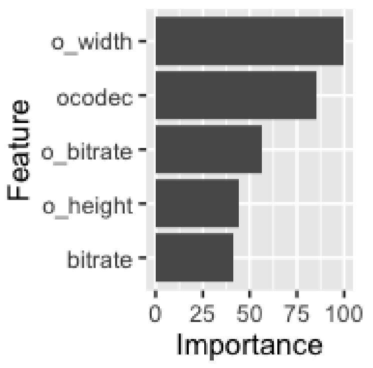
Table 3.4 – Results of model performance on the complete dataset

on the other hand, seems to have faltered on the test dataset as its R^2 value decreased slightly and the RMSE values increased by a tiny bit. In terms of RMSE however, The NN model comes out on top again with an RMSE value of 4.84.

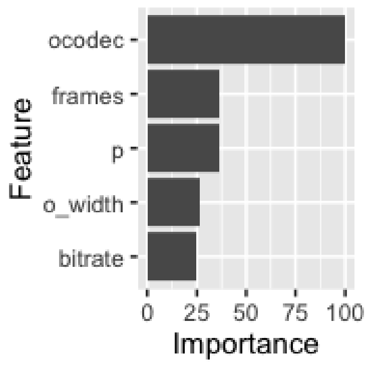
3.6.2 Variable Importance



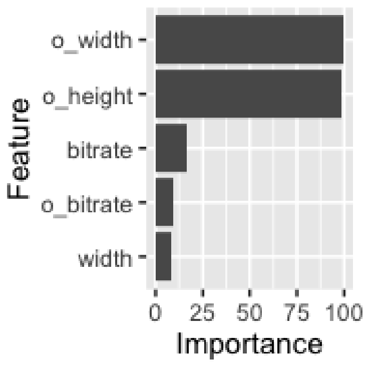
(a) avgNNet Important Predictors



(b) MARS Important Predictors



(a) NNet Important Predictors



(b) K-Nearest neighbors important predictors

Figure 3.12 – Top Predictors for Each Model

This section intends to show the importance of the non-applicability of linear models to complex datasets like the youtube dataset where the number of predictors are very high with complex inter-predictor relationships.

4 Conclusions and Recommendations

4.1 Conclusions

The main objective of this report was to offer a machine learning trained predictive model for video transcoding rates given a few video input and output parameters. Section 3 uses a clear, sequential method that seeks to mimic standard regressive analysis techniques for high dimensional datasets including feature selection, variable importance and model cross-validation. The final predictive model is to able to correctly predict transcoding rates given input and output video standards, and a few other video characteristics. The predictive ability of this model was quantified by using R^2 and RMSE, with the neural network model coming out on top of all other models in its ability to predict transcoding rates for the test dataset.

It is recommended that one use the neural network model file available at the Github repository for this report, cited here [23]. This file maybe loaded into an R environment using the $load(filename)$ function call and new datasets maybe used to gain predictions for new samples provided the samples come in a pre-configured vector with multiple features where each feature corresponds to certain video characteristics like codec type, bitrate, etc. It is recommended that the technical user seeking to use this model use the $head(processPredictors_training)$ R command to get an exact format for the new dataset's columns and rows.

4.2 Recommendations

It is recommended that the models be re-trained using additional computational power and a much bigger portion of the youtube dataset. Even though the models generated got a pretty good fit, the amount of time taken to train each model was extremely high for the hardware available for the author, a paltry Macbook Air 13" with no GPU processing power. It is also recommended that the youtube dataset be used to train a Support Vector Machine dataset in hopes of gaining a better fitting model. An additional augmentation that would be very useful for distributed video transcoding operations is to add a constraint variable that seeks to emulate the processing capacities of different server farms, including other sub-constraints such as network congestion, and isolation.

A Appendix

A.1 Programming Code

All the code used to generate the visualizations for this report can be found in the github repository located at Github.

A.1.1 Model Training and Results Visualization - models.R

```
# Multiple training routines EPage 82
save_plots = T
library(caret)
library(AppliedPredictiveModeling)
library(dplyr)
library(lattice)
set.seed(0)
#load('chemproc.dat')
library(doMC)
registerDoMC(cores = 4)

saveModel <- function(varN) {
    varStr = deparse(substitute(varN))
    save(varN, file=varStr)
}
tm_all = read.table('transcoding_mesurment.tsv', sep='\t',
 ↪    header=TRUE)
# data(ChemicalManufacturingProcess)
transc_meas = sample_frac(tm_all, 0.05)
transc_meas\$icodec = as.integer(transc_meas\$codec)
transc_meas\$ocodec = as.integer(transc_meas\$o_codec)
transc_meas\$utime =
 ↪    transc_meas\$frames/(transc_meas\$utime)
predArray = array(c(4:7, 11, 23:24))
processPredictors = transc_meas[, predArray]
yield = transc_meas[, 22]
## read all data without sampling
tm_all\$icodec = as.integer(tm_all\$codec)
tm_all\$ocodec = as.integer(tm_all\$o_codec)
tm_all\$utime = tm_all\$frames/(tm_all\$utime)
pp_all = tm_all[, predArray]
yield_all = tm_all[, 22]
# Look for any features with no variance:
zero_cols = nearZeroVar( processPredictors )
print( sprintf("Found %d zero variance columns from
 ↪    %d",length(zero_cols), dim(processPredictors)[2] ) )
processPredictors = processPredictors[,-zero_cols] # drop
 ↪    these zero variance columns
pp_all = pp_all[, -zero_cols]
# Split this data into training and testing sets:
#
training = createDataPartition( yield, p=0.8 )
processPredictors_training =
 ↪    processPredictors[training\$Resample1,]
yield_training = yield[training\$Resample1]
processPredictors_testing =
 ↪    processPredictors[-training\$Resample1,]
yield_testing = yield[-training\$Resample1]
# Build various nonlinear models and then compare
 ↪    performance:
preProc_Arguments = c("center","scale")
fitControl = trainControl(method = "repeatedcv", #10-fold
 ↪    cross validation
                          number=10,repeats = 10,
                           ↪    returnResamp = "all")
# A K-NN model:
#
set.seed(10)
knnModel = train(x=processPredictors_training,
 ↪    y=yield_training, method="knn",
 ↪    preProc=preProc_Arguments, tuneLength=40, trControl =
 ↪    fitControl)
saveModel(knnModel)

# predict on training/testing sets
knnPred = predict(knnModel,
 ↪    newdata=processPredictors_training)
knnPR = postResample(pred=knnPred, obs=yield_training)
rmses_training = c(knnPR[1])
```

```
r2s_training = c(knnPR[2])
methods = c("KNN")
## try to predict all of the dataset now since our sampled
 ↪    data was only 5%
knnPred_all = predict(knnModel, newdata=pp_all)
knnPR_all = postResample(pred=knnPred_all, obs = yield_all)
rmses_testing = c(knnPR_all[1])
r2s_testing = c(knnPR_all[2])

knnPredVals = extractPrediction(list(knnModel),
 ↪    testX=processPredictors_testing, testY=yield_testing,
 ↪    unkX=pp_all)
trellis.device(device="png", width=8, height=5, units="in",
 ↪    filename="../wrkreport/images/knn_pred_obs.png",
 ↪    res=100)
print(plotObsVsPred(knnPredVals))
dev.off()
ggplot(knnModel, knnModel\$metric[1]) +ggtitle(" Performance
 ↪    of k-nearest neighbors model")
ggsave("../wrkreport/images/knnresults.png", width=8,
 ↪    height=5, units="in", dpi=100)

# A Neural Network model:
#
nnGrid = expand.grid( .decay=c(0,0.01,0.1), .size=1:10,
 ↪    .bag=FALSE )
set.seed(0)
nnetModel = train(x=processPredictors_training,
 ↪    y=yield_training, method="nnet", trControl =
 ↪    fitControl, preProc=preProc_Arguments,
 ↪    linout=TRUE,trace=FALSE,MaxNWts=10 *
 ↪    (ncol(processPredictors_training)+1) + 10 + 1,
 ↪    maxit=500)
saveModel(nnetModel)

nnetPred = predict(nnetModel,
 ↪    newdata=processPredictors_training)
nnetPR = postResample(pred=nnetPred, obs=yield_training)
rmses_training = c(rmses_training,nnetPR[1])
r2s_training = c(r2s_training,nnetPR[2])
methods = c(methods,"NN")
## try to predict all of the dataset now since our sampled
 ↪    data was only 5%
nnetPred_all = predict(nnetModel, newdata=pp_all)
nnetPR_all = postResample(pred=nnetPred_all, obs =
 ↪    yield_all)
rmses_testing = c(rmses_testing, nnetPR_all[1])
r2s_testing = c(r2s_testing, nnetPR_all[2])

nnetPredVals = extractPrediction(list(nnetModel),
 ↪    testX=processPredictors_testing, testY=yield_testing,
 ↪    unkX=pp_all)
trellis.device(device="png", width=8, height=5, units="in",
 ↪    filename="../wrkreport/images/nnet_pred_obs.png",
 ↪    res=100)
print(plotObsVsPred(nnetPredVals))
dev.off()
quartz()
ggplot(nnetModel, metric="RMSE") + ggtitle("Performance of
 ↪    neural network model")
ggsave("../wrkreport/images/nnetresults.png", width=8,
 ↪    height=5, units="in", dpi=100)

# Averaged Neural Network models:
#
set.seed(45)
avNNetModel = train(x=processPredictors_training,
 ↪    y=yield_training, trControl = fitControl,
 ↪    method="avNNet", preProc=preProc_Arguments,
 ↪    linout=TRUE,trace=FALSE,MaxNWts=10 *
 ↪    (ncol(processPredictors_training)+1) + 10 + 1,
 ↪    maxit=500)
saveModel(avNNetModel)
avNNetPred = predict(avNNetModel,
 ↪    newdata=processPredictors_training)
avNNetPR = postResample(pred=avNNetPred, obs=yield_training)
rmses_training = c(rmses_training,avNNetPR[1])
r2s_training = c(r2s_training,avNNetPR[2])
methods = c(methods,"AvgNN")
## try to predict all of the dataset now since our sampled
 ↪    data was only 5%
avnnetPred_all = predict(avNNetModel, newdata=pp_all)
avnnetPR_all = postResample(pred=avnnetPred_all, obs =
 ↪    yield_all)
rmses_testing = c(rmses_testing, avnnetPR_all[1])
r2s_testing = c(r2s_testing, avnnetPR_all[2])
avnnetPredVals = extractPrediction(list(avNNetModel),
 ↪    testX=processPredictors_testing, testY=yield_testing,
 ↪    unkX=pp_all)
trellis.device(device="png", width=8, height=5, units="in",
 ↪    filename="../wrkreport/images/avnnet_pred_obs.png",
 ↪    res=100)
print(plotObsVsPred(avnnetPredVals))
dev.off()
ggplot(avNNetModel, metric="RMSE") + ggtitle("Performance of
 ↪    average neural network model")
ggsave("../wrkreport/images/avnnetresults.png", width=8,
 ↪    height=5, units="in", dpi=100)

## MARS model:
marsGrid = expand.grid(.degree=1:2, .nprune=2:38)
set.seed(0)
marsModel = train(x=processPredictors_training,
 ↪    y=yield_training, method="earth",  trControl =
 ↪    fitControl, preProc=preProc_Arguments,
 ↪    tuneGrid=marsGrid)
saveModel(marsModel)
marsPred = predict(marsModel,
 ↪    newdata=processPredictors_training)
marsPR = postResample(pred=marsPred, obs=yield_training)
rmses_training = c(rmses_training,marsPR[1])
r2s_training = c(r2s_training,marsPR[2])
methods = c(methods,"MARS")
## try to predict all of the dataset now since our sampled
 ↪    data was only 5%
```

```
marsPred_all = predict(marsModel, newdata=pp_all)
marsPR_all = postResample(pred=marsPred_all, obs =
 ↪    yield_all)
rmses_testing = c(rmses_testing, marsPR_all[1])
r2s_testing = c(r2s_testing, marsPR_all[2])
marsPredVals = extractPrediction(list(marsModel),
 ↪    testX=processPredictors_testing, testY=yield_testing,
 ↪    unkX=pp_all)
trellis.device(device="png", width=8, height=5, units="in",
 ↪    filename="../wrkreport/images/mars_pred_obs.png",
 ↪    res=100)
plotObsVsPred(marsPredVals)
dev.off()
quartz()
ggplot(marsModel, metric="RMSE") + ggtitle("Performance of
 ↪    MARS model")
ggsave("../wrkreport/images/marsresults.png", width=8,
 ↪    height=5, units="in", dpi=100)

# Lets see what variables are most important in the MARS
 ↪    model:
ggplot(varImp(nnetModel), top=5);
ggsave("../wrkreport/images/nnetvarimp.png", width=5,
 ↪    height=5, units="cm", dpi=100)
ggplot(varImp(knnModel), top=5);
ggsave("../wrkreport/images/knnvarimp.png", width=5,
 ↪    height=5, units="cm", dpi=100)
ggplot(varImp(avNNetModel), top=5);
ggsave("../wrkreport/images/avnnetvarimp.png", width=5,
 ↪    height=5, units="cm", dpi=100)
ggplot(varImp(marsModel), top=5);
ggsave("../wrkreport/images/marsvarimp.png", width=5,
 ↪    height=5, units="cm", dpi=100)
# Package the results up:
res_training = data.frame( rmse=rmses_training,
 ↪    r2=r2s_training )
rownames(res_training) = methods
training_order = order( -res_training\$rmse )
res_training = res_training[ training_order, ]
library(Hmisc)
latex(res_training, file="../wrkreport/train_results.tex")
res_testing = data.frame( rmse=rmses_testing, r2=r2s_testing
 ↪    )
rownames(res_testing) = methods
res_testing = res_testing[ training_order, ]
latex(res_testing, file="../wrkreport/testing_results.tex")
# EPage 82
resamp = resamples(
 ↪    list(knn=knnModel,mars=marsModel,nnet=nnetModel,avnnet=avNNetModel)
 ↪    )
resamp_sum = summary(resamp)
latex(resamp_sum\$statistics, file =
 ↪    "../wrkreport/resamp_stats.tex")
trellis.device(device="png", width=8, height=5, units="in",
 ↪    filename="../wrkreport/images/resamp_bwplot.png",
 ↪    res=100)
bwplot(resamp, scales="free")
dev.off()
trellis.device(device="png", width=8, height=5, units="in",
 ↪    filename="../wrkreport/images/resamp_dotplot.png",
 ↪    res=100)
dotplot(resamp, scales="free")
dev.off()
trellis.device(device="png", width=8, height=5, units="in",
 ↪    filename="../wrkreport/images/resamp_parallelplot.png",
 ↪    res=100)
parallelplot(resamp, metric="RMSE")
dev.off()
print( summary(diff(resamp))
save(file="chemproc.dat", list=ls())
```

A.2 Youtube Video Dataset Characteristics

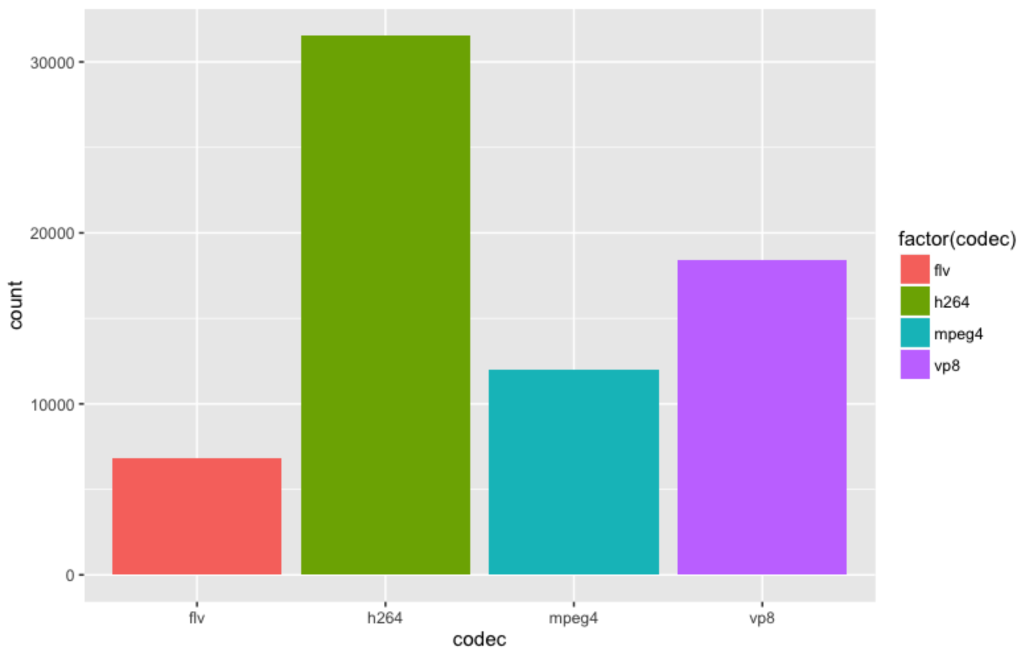


Figure A.1 – Histogram of videos by codec type in the youtube dataset

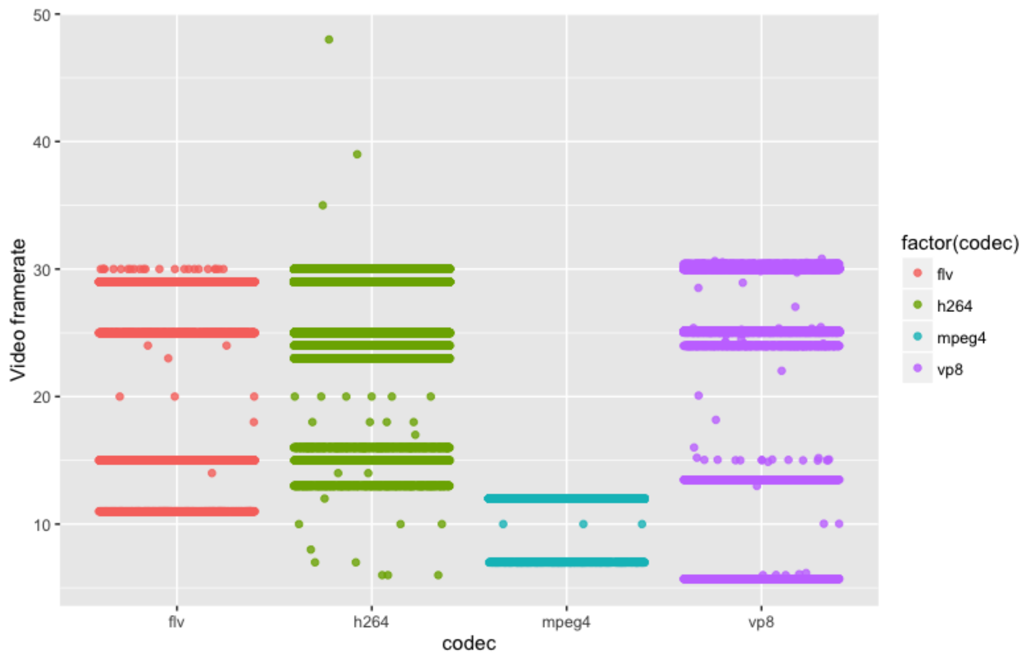


Figure A.2 – Jitter plots of framerates separated by codec type

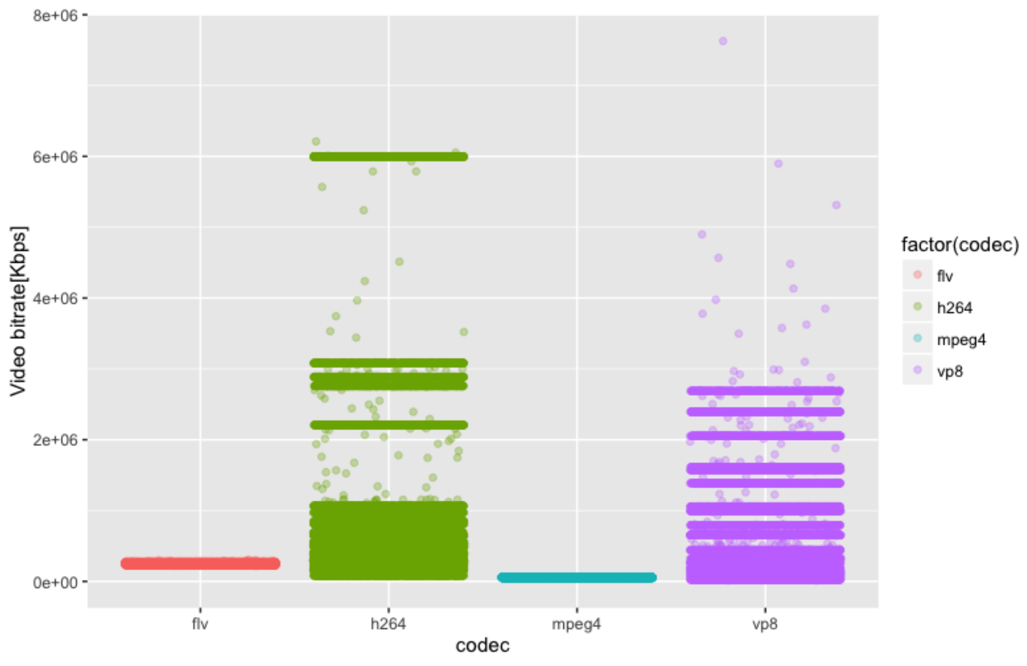


Figure A.3 – Jitter plots of bitrates separated by codec type

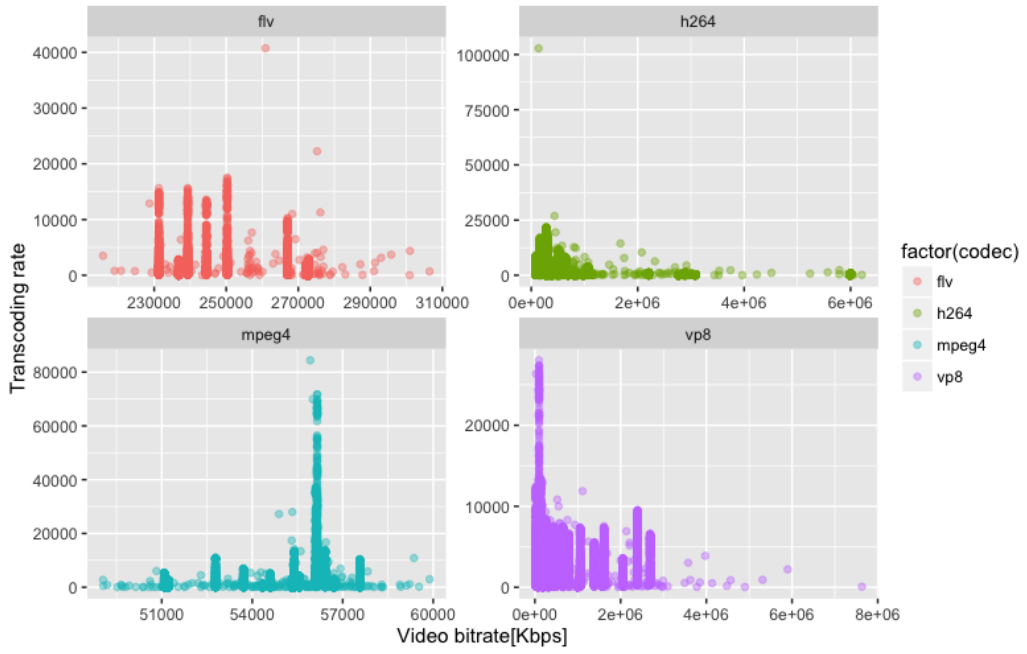


Figure A.4 – Transcoding rate [fps] v/s video bitrate [Kbps]

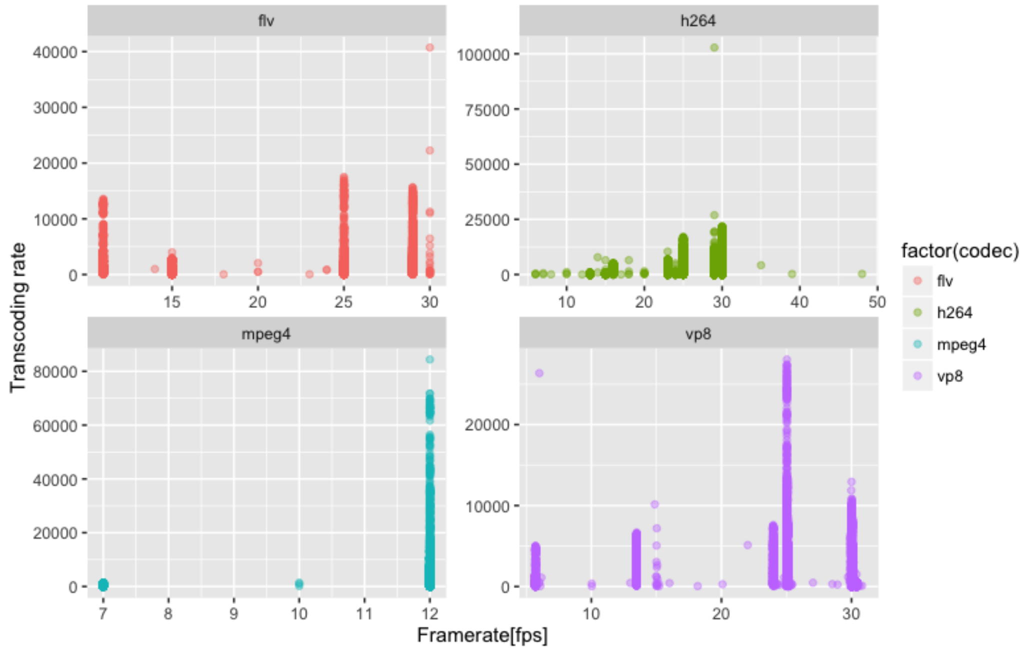


Figure A.5 – Transcoding rate [fps] v/s video framerate [fps]

References

[1] D. Kundur, "Introduction to Video Processing." [Online]. Available: http://www.comm.utoronto.ca/~dkundur/course_info/real-time-DSP/notes/13_Kundur_Intro_Video_Edge_Detection.pdf

[2] R. Choupani, S. Wong, and M. Tolun, "Video coding and transcoding: A review," 2007.

[3] "Video coding with H.264/AVC: Tools, performance, and complexity," *IEEE Circuits and Systems Magazine*, vol. 4, no. 1, pp. 7–28, 2004.

[4] M. Kuhn, "Model training and tuning," http://topepo.github.io/caret/training.html, (Visited on 01/11/2016).

[5] "What my deep model doesn't know... — yarin gal - blog — cambridge machine learning group," http://mlg.eng.cam.ac.uk/yarin/blog_3d801aa532c1ce.html, (Visited on 01/11/2016).

[6] "Ffmpeg - open source audio video processing library," https://www.ffmpeg.org/, (Visited on 01/04/2016).

[7] C. Holder, "VIDEO TRANSCODING USING MACHINE LEARNING," no. December, 2008.

[8] S. Lefèvre, J. Holler, and N. Vincent, "A review of real-time segmentation of uncompressed video sequences for content-based search and retrieval," *Real-Time Imaging*, vol. 9, no. 1, pp. 73–98, 2003.

[9] A. Saggi, "A framework for multimedia playback and analysis of MPEG-2 videos with FFmpeg," Ph.D. dissertation, 2010.

[10] T. Wiegand, "Overview of the H. 264/AVC video coding standard," *. . . and Systems for Video . . .*, vol. 13, no. 7, pp. 560 –576, 2003. [Online]. Available: http://ieeexplore.ieee.org/ielx5/76/27384/01218189.pdf?tp={&}arnumber=1218189{&}isnumber=27384$\delimiter"026E30F$nhttp://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=1218189

[11] "Videolan - x264, the best h.264/avc encoder," http://www.videolan.org/developers/x264.html, (Visited on 01/07/2016).

[12] "Transcoding best practices - jwplayer," http://www.jwplayer.com/blog/transcoding-best-practices/, (Visited on 01/07/2016).

[13] "Uci machine learning repository: Online video characteristics and transcoding time dataset data set," https://archive.ics.uci.edu/ml/datasets/Online+Video+Characteristics+and+Transcoding+Time+Dataset, (Visited on 01/11/2016).

[14] M. Kuhn and K. Johnson, "Appliedpredictivemodeling: Functions and data sets for 'applied predictive modeling'," 2014, r package version 1.1-6. [Online]. Available: http://CRAN.R-project.org/package=AppliedPredictiveModeling

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[16] M. K. C. from Jed Wing, S. Weston, A. Williams, C. Keefer, A. Engelhardt, T. Cooper, Z. Mayer, B. Kenkel, the R Core Team and Michael Benesty, R. Lescarbeau, A. Ziem, L. Scrucca, Y. Tang, and C. Candan., *caret: Classification and Regression Training*, 2015, r package version 6.0-62. [Online]. Available: http://CRAN.R-project.org/package=caret

[17] D. Sarkar, *Lattice: Multivariate Data Visualization with R*. New York: Springer, 2008, iSBN 978-0-387-75968-5. [Online]. Available: http://lmdvr.r-forge.r-project.org

[18] H. Wickham, *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. [Online]. Available: http://had.co.nz/ggplot2/book

[19] M. Kuhn, "Visualizations," http://topepo.github.io/caret/visualizations.html, (Visited on 01/11/2016).

[20] "Knn with categorical variables," http://faculty.nps.edu/sebuttre/home/Research/KnnCat/ordsdoc.html, (Visited on 01/11/2016).

[21] M. Kuhn, "Building Predictive Models in R Using the caret Package," *Journal Of Statistical Software*, vol. 28, no. 5, pp. 1–26, 2008. [Online]. Available: http://www.jstatsoft.org/v28/i05/

[22] J. H. Friedman, "Multivariate adaptive regression splines," *The annals of statistics*, pp. 1–67, 1991.

[23] P. Kandarpa, "prajnak/machine_learning_experiments," https://github.com/prajnak/machine_learning_experiments, (Visited on 01/11/2016).