# Using a convolutional neural network to classify levels of left ventricular ejection fraction from echocardiography reports: A baseline study

Billy Wu

# Abstract

This study applied a deep learning model to classify levels of left ventricular ejection fraction using free-text data recorded in 965 echocardiography (ECHO) reports from Auckland City and Middlemore Hospitals. Currently, clinicians are required to manually review a patient’s ECHO report to determine their level of left ventricular ejection fraction (LVEF). While several efforts have been made to automatically extract information from ECHO reports, none have adopted a deep learning approach. This study aimed to establish some baseline performance measures using a 1-dimensional convolutional neural network (CNN) and determine strategies for improving classification precision. The results were promising but were unable to exceed precision or F-scores of 80% for *mild*, *moderate*, or *severe* LVEF; irrespective of how the training data was sampled, how deeply connected the neural network layers were, or well the model was tuned. This study strongly supports the need to develop a chunking algorithm that could help captures conceptual, contextual, and measurement features from ECHO reports.

Keywords: *natural language processing*, *text classification*, *ejection fraction*

# Introduction

Left ventricular ejection fraction (LVEF) is a diagnostic measure of overall heart health and one of the most important prognostic indicators for people with cardiovascular diseases (1). The ejection fraction is represented by the percentage of blood ejected from the left ventricle (LV) with each systole. Individuals with values of LVEF 50 to 70% and above are considered to have *normal* functioning, while individuals with values less than 45% are considered at higher cardiovascular risk and mortality (2). Notably, while LVEF values between 50 and 55% are classified as *normal* functioning, this band is considered *borderline* in the literature (3). There is evidence supporting the association between reduced LV functioning, heart failure with acute coronary syndrome, and an adverse prognosis. Significant improvements in LVEF can be achieved through treatment for patients with heart failure who have LVEF values below 40% (4). These include the use of of angiotensin converting enzyme inhibitors (ACEIs), angiotensin receptor blockers (ARBs), beta-blockers and mineralocorticoid receptor antagonists (MRAs).

The evaluation of LVEF come from imaging the heart – of which – there are several modalities including echocardiography (ECHO), magnetic resonance imagining (MRI), computed tomography (CT), and radionuclicide angiography. Notably, ejection fraction is typically measured only on the left ventricle in echocardiography. The information from ECHO reports that pertain to the LV are represented as either unstructured or semi-structured data. It would be necessary for a clinician to manually review a patient’s ECHO reports in order to determine their level of LVEF (e.g. LVEF of 30% or less). As such, while estimates from ECHO reports have shown to be reliable, there is a degree of dependency on the clinicians to manually assess LV functioning in most ECHO reports. However, there are several challenges in extracting LVEF from ECHO reports given its free-text format. These include i) information presented in a variety manners and styles; ii) information located in different parts of the report; iii) information presented in a mix of quantitative and qualitive formats; and iv) a mixture of LVEF labelling as qualitative descriptors (5).

## Research Rationale

An intelligent, reliable, and systematic method for extracting ejection fraction values from ECHO reports would benefit patients, clinicians, and the wider health system. For patients with heart failure, it is vital for clinicians to obtain initial outpatient cardiology evaluations in a timely manner. Manually processing ECHO reports is time consuming and costly to the health system, while rule-based data extraction methods can lack in generalisability, accuracy, and reliability. Therefore, this study takes the opportunity to utilise advanced methods for accurately stratifying levels of LVEF from ECHO reports; to improve the efficiency of the clinician decision-making process based on cardiovascular risk. This study will explore the use of natural language processing (NLP) for text classification of LVEF using free-text data from ECHO reports.

# Literature Review

Several efforts have been made to automatically extract information from ECHO reports. Garvin et al. (2012) built an NLP processing system to extract LVEF from 765 ECHO reports using a binary classification of <40% (i.e. reduced LVEF) and >= 40% (i.e. not reduced LVEF) (6). The labels were a “gold standard” as they were annotated by two independent reviewers with a third reviewer who adjudicated any disagreements. The researchers presented a rules-based classifier model and achieved a performance of 95% precision, 89% recall, and 92% F-score. While these results were impressive, a rules-based model would not adapt well to new information. Furthermore, reduced LVEF can be further classified as *moderate* (35 - 39%) or *severe* (<35%); which the model presented by Garvin et al. does not distinguish.

Kim et al. (2015) presented an NLP system to classify the contextual use of both quantitative and qualitative LVEF for both ECHO and radiography reports; to determine one of five contextual categories: *summary*, *interpretation*, *technical measurement*, *recommendation*, *past findings* (7). The authors approached the task as a topic classification problem and created a support vector machine (SVM) classifier to determine each of the five contextual categories. As a supervised learning problem, clinical notes were manually annotated with corresponding contextual labels. The samples were divided into qualitative and quantitative assessments to capture both words and numbers of interest. For example, a *summary* labels could be “Normal LV function” as a qualitative phrase or “EF 55-65%” as a quantitative phrase. The most impressive aspect of the study was the union of different features that feed into the multi-class SVM classifier which included lexical (BOW) features, related concept features (n-grams on either side of a mention of “*LVEF*”), and document section features (titles relating to the contextual keywords). Overall, the classification F-score was an impressive 96% for quantitative phrases and 94% for qualitative phrases.

In a continuation of their research, Kim et al. (2017) applied their 2015 model to a new dataset in order to i) test the generalisability of detecting *concept mentions* (i.e. terms that represent “LVEF”) and the associated values that relate to each *concept mention*, and to ii) update and improve the existing model using the new dataset (8). The updated models performed well, reaching a precision of 96.5% for *mentions of LVEF* (e.g. “visual estimate of ejection fraction”), 87.8% for *qualitative assessments of LVEF* (e.g. “Normal” or “Severe”), and 93.0% for *quantitative values of LVEF* (e.g. “0.45” or “50%”). Notably, the updated model incorporated BIO tags into its feature set, along with n-grams of words, part-of-speech (POS) tags, n-grams of POS tags, word morphology, word shape information, and skip-grams.

Patterson et al. (2017) developed an NLP system with very high precision without the use of sophisticated machine learning methods – rather, the researchers used custom dictionaries, rules, and regular expressions to identify concept terms, values, and units; to extract heart function measurements recorded in ECHO reports, radiology reports, and general clinical notes (9). The system achieved a precision of 98.2% on ECHO reports, 98.2% on radiology reports, and 93.6% on general clinical notes, for detecting overall concept terms and values. Specifically, the detection of LVEF achieved a precision ranging between 96.8% and 100% and a recall of 80.1%. The most notable design element that the researchers implemented was the system’s ability to identify *concepts mentions* (e.g. terms representing “LVEF”). Patterson et al. pointed out that *concept identification* relies on a comprehensive lexicon that capture a wide range of terms associated with a target concept (e.g. “ejection fraction”). While standardised medical vocabularies such as the Unified Medical Language System (UMLS) enable rudimentary or common terms to be captured (e.g. “left ventricular ejection fraction”), any locally defined abbreviations (e.g. “EF”) , acronyms (e.g. “HF-rEF”), and spelling variations or misspellings, will be limited and unavailable in the standard dictionary (9).

Xie et al. (2017) developed an LVEF extraction algorithm from over 600,000 ECHO reports, of which 70% contained numeric EF values while 30% contained only qualitative description that indicated the left ventricular function (10). Rather than building a classifier to determine the level LV functioning, the researchers aimed to classify whether an LVEF text description (either with or without quantitative values) could be accurately drawn from an ECHO report. Therefore, the purpose of the clinician’s gold standard is to validate whether specific text descriptions - used to determine a person’s level of LV impairment - could be accurately identified by the model compared to manually reviewing each ECHO report. To some extent, the information retrieval process presented by Xie et al. was akin to that of Kim et al. and Patterson et al.; whereby documents were segmented into sections and phrases that involved calling upon keywords and their relationship with qualitative and quantitative values.

## Research objective

The directional aim of this study is to provide cardiologists with a proof-of-concept in determining the feasibility and potential utility of a learning algorithm for classifying levels of ejection fraction in echocardiography reports. The objective is to establish baseline performance measures for classifying levels of LV functioning and to determine strategies for improving classification accuracy. The key tasks involved in this study include:

* Establishing baseline performances using off-the-shelf classifier models
* Exploring different feature sets as input to a classifier model
* Establishing baseline performances using a tuned CNN sequential model
* Determining strategies for improving classification accuracy

# Data Description & Preliminary Analysis

## Data Source

The data utilised in this study came from MENZACS; the *Multi-Ethnic New Zealand Study of Acute Coronary Syndromes*; a case-control study funded and supported by the Heart Health Research Trust of the Heart Foundation, the Healthier Lives National Science Challenge and the University of Auckland. The goal of MENZACS is to better understand factors that contribute toward a high risk of developing heart disease. Beginning in 2015, the MENZACS study began recruiting patients who were presenting to hospital for the first time with acute coronary syndrome. The study has currently recruited around 2,000 patients from Auckland City Hospital, Middlemore Hospital, Waikato Hospital, and Christchurch Hospital. Patients in this study have given consent for access to their clinical record and the study has been approved by the New Zealand Health and Disability Ethics Committee (15/NTB/59).

In this study, a sample of 987 ECHO reports were obtained from 391 patients at Auckland City Hospital and 372 patients from Middlemore Hospital. Each echocardiogram report was reviewed by a cardiologist who determine the level of LVEF which were classified into *normal*, *mild*, *moderate*, and *severe*. These levels represent the clinician’s coding and the study’s gold-standard labelling. Samples were excluded if a report had either no ejection fraction or had an unclassifiable ejection fraction. After the exclusion, the study had 965 echocardiogram reports, each labelled with a cardiologist’s clinical coding for LVEF.

Each echocardiogram report contains a number a free-text fields. In this study, the text fields ‘Conclusion’ and ‘LV Comments’ were used as the textual data. Cardiologist’s gold standard – clinician coding – is provided alongside each report. For example:

|  |  |
| --- | --- |
| **Clinician Coding** | **Conclusions / Comments** |
| Normal | *Right heart size and right ventricular (RV) systolic function are normal. An accurate Doppler estimate of pulmonary artery systolic pressure could not be obtained. The tricuspid valve is structurally unremarkable. Trivial tricuspid regurgitation (TR). The pulmonary valve and proximal pulmonary arteries are unremarkable. The atrial septum appears intact from the views obtained.* |
| Mild | *Atrial fibrillation - heart rate generally high. Ventricular ectopic beats noted. From 2-D measurement the left ventricle (LV) is moderately dilated. LV ejection fraction is difficult to estimate (high heart rate, AF with ectopy) but appears mildly reduced with regional variation (hypokinesis of the basal inferior septum and the mid to distal inferior and lateral walls). There is subjectively moderate concentric LV hypertrophy (LVH).* |
| Moderate | *Sinus rhythm with frequent ventricular ectopy. Mildly dilated left ventricle with moderate segmental systolic dysfunction. Regional wall motion abnormalities consistent with a large anteroapical infarct. Estimated LVEF 35%. Moderate diastolic dysfunction. Mild left atrial dilatation. Normal right ventricular size and systolic function. Compared with the study performed 12/07/2015, there is no significant change in left ventricular systolic function.* |

## Pre-Processing

The echocardiogram reports were exported from hospital databases to excel spreadsheets. The free-text field “Conclusions” exist on all reports and were used as the primary source of textual data. The field “Comments” were concatenated as additional textual data – if available. This concatenation process added more syntactic richness to the study data; and expanded the dictionary and features available to model building. In preparation for model training and testing, several cleaning steps were deployed to tighten the study data:

1. Removal of special character, digits, newline, duplicate spaces, and accented characters
2. Removal of stop-word removal excluding ‘no’ and ‘not’
3. Contraction Expansion
4. Lemmatisation
5. Lower-casing

Using the example above for ‘severe’, the output from pre-processing is as follows:

|  |  |
| --- | --- |
| Severe  (Original) | *1. Normal LV size with moderate to severe systolic impairment. 2. Moderate RV impairment. Note follow up echo would best with echo contrast. All LV segments are abnormal to some degree. The apex is mildly expanded. Overall there is severe LV systolic impairment with estimated LVEF ~25 - 30%. Unable to exclude LV thrombus as the apex is not well seen.* |
| Severe (Processed) | *normal lv size moderate severe systolic impairment moderate rv impairment note follow echo would best echo contrast lv segment abnormal degree apex mildly expand overall severe lv systolic impairment estimate lvef unable exclude lv thrombus apex not well see* |

## Data Description

At end of pre-processing, the complete dataset consisted of a total of 70,861 words; across 965 samples. Samples ranged between 7 and 199 words with an average of 73 words. Figure 1 shows the sample length distribution across the dataset which was used to inform a maximum length at 175 words in the subsequent word vectorisation process. Figure 2 shows the top words appearing in the dataset. The top 6 words were ‘*LV’* (the abbreviation for left ventricle), ‘*normal’*, ‘*size’, ‘mild’, ‘wall’,* and *‘no’*. As aforementioned, the words ‘no’ and ‘not’ were untouched by stop-word removal (i.e. retained in the corpus) as they potentially contribute toward important n-grams.

Figure 3 shows all available labels in our dataset. Each sample was classified by a cardiologist with respect to *normal*, *mild*, *moderate*, or *severe* LVEF. In a data set of 965 samples, there were 551 *normal* (57%), 222 *mild* (23%), 126 *moderate* (13%), and 66 *severe* (7%). This distribution is imbalanced with *mild*, *moderate*, and *severe* existing as minority classes. The grouping of *moderate* and *severe* was considered as this created a class containing 192 samples (20%).

|  |  |
| --- | --- |
|  | Figure 1 - Distribution of word lengths per sample in the dataset |
|  | Figure 2 - Frequency of most common occurring words in the dataset |
|  | Figure 3 - Number of samples containing each clinician-coded label; normal, mild, moderate, and severe. |

## Baseline Performance

Initial performance measures using standard machine learning models were established using the Python machine learning library, *Scikit-learn*. This baseline performance provided a reference for comparison with more sophisticated learning models. It is expected that most advanced methods would perform better than an untuned Naïve Bayes classifier. The models included in establishing a baseline performance were: i) *Naïve Bayes*, ii) *Logistic Regression*, iii) *Random Forest*, and iv) *Multilayer Perceptron (MLP)*. These models were selected to represent an eclectic mix of simplicity, effectiveness, ensemble learning, and neural network learning. Furthermore, each of these models are inherently multi-class and can classify multiple output labels. The dataset was split into 80% training and 20% testing, and text for each sample from the training set were vectorised using a Bag-of-words (BOW) model. Two sparse matrices were created, each with 1501 features; with the training set containing 772 samples and the testing set containing 193 samples.

Machine learning algorithms were applied in two ways – first as multi-level classifiers. Table 1 shows each of the model’s performance when trained as a multi-level classifier. The minority classes performed poorly as they struggled to be distinguished from the other classes. Indeed, precision, recall and F1-score were not ideal for any given class other than *normal*. Overall, the MLP model performed better than the rest, given the same training and testing datasets.

Table 1 - Baseline PRF test performance for each model trained as a multi-level classifier

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Class Label** | **Precision** | **Recall** | **F1-score** |
| Naïve Bayes | 0.7254 | Normal | 0.91 | 0.90 | 0.90 |
| Mild | 0.33 | 0.37 | 0.35 |
| Moderate | 0.42 | 0.37 | 0.39 |
| Severe | 0.55 | 0.75 | 0.63 |
| Logistic Regression | 0.8031 | Normal | 0.96 | 0.92 | 0.94 |
| Mild | 0.55 | 0.55 | 0.55 |
| Moderate | 0.50 | 0.57 | 0.53 |
| Severe | 0.55 | 0.67 | 0.60 |
| Random Forest | 0.7720 | Normal | 0.93 | 0.89 | 0.91 |
| Mild | 0.55 | 0.46 | 0.50 |
| Moderate | 0.46 | 0.63 | 0.53 |
| Severe | 0.45 | 0.71 | 0.56 |
| MLP | 0.7927 | Normal | 0.92 | 0.90 | 0.91 |
| Mild | 0.61 | 0.56 | 0.58 |
| Moderate | 0.54 | 0.61 | 0.57 |
| Severe | 0.55 | 0.75 | 0.63 |

The second approach was to apply each of the models as a binary classifier. Unsurprisingly, the Logistic Regression model performed slightly better than the MLP model. However, the MLP model offered a more practical baseline for comparison as the objective of this study was to train a convolutional neural network for binary classification. Furthermore, the main interest in binary classification is the ability to accuracy detect the minority class. As such, the most important measure in this study was precision; starting with a baseline of 89% for *normal*, 39% for *mild*, 54% for *moderate*, and 55% of *severe*.

Table 2 - Baseline PRF test performance for each model trained as a multi-level classifier

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Class** | **Accuracy** | **Class Label** | **Precision** | **Recall** | **F1-score** |
| Multilayer Perceptron | Normal | 0.8808 | No = 0 | 0.86 | 0.82 | 0.84 |
| Yes = 1 | **0.89** | 0.92 | **0.91** |
| Mild | 0.8394 | No = 0 | 0.93 | 0.88 | 0.91 |
| Yes = 1 | **0.39** | 0.54 | **0.46** |
| Moderate | 0.9067 | No = 0 | 0.96 | 0.93 | 0.95 |
| Yes = 1 | **0.54** | 0.70 | **0.61** |
| Severe | 0.9637 | No = 0 | 0.99 | 0.97 | 0.98 |
| Yes = 1 | **0.55** | 0.75 | **0.63** |

# Research Design

## Word Embedding

In previous baseline performance, a simple BOW model was used to vectorise a whole sequence of text and represented them as a single feature vector. For the CNN design, word embeddings were used which represents each word as a word vector. In this study, two methods of word embedding were experimented in the Keras embedding layer: 1) Simple word embedding as CNN input layer, and 2) Pre-trained word embedding matrix as weights.

The first approach was to create a Keras embedding layer based the dataset’s vocabulary. This embedding layer is the first layer of the CNN. This process begins with tokenisation using Tokenizer from *Scikit-learn*. In the first instance, an internal dictionary is created containing a list of texts from the dataset. Each word in the dictionary is assigned an integer value based on its frequency (where lower values indicate higher frequency). Next, the text from each sample in the dataset is transformed to a sequence of integers which corresponds to the value from the internal dictionary. In Keras CNN model, the embedding layer takes all the integers and maps them to a dense vector.

## Feature Engineering

The first component of the feature vector is the sequence of integers that correspond to values in the internal library. This effectively treats each word of the sample as a feature in the embedding layer (i.e. as a lexical feature). For a training set consisting of 772 sample, an n-dimensional array with a shape of 772 and 175 is created. In addition, this study experimented with additional features that were concatenated to this array. In doing so, the feature vector grows longer but the length of the total matrix remains the same (i.e. 772). In this study, the following additional features were added to the feature set:

* **TF-IDF features at word level**: adds a matrix of TF-IDF scores applied to every term in the sample (+1621 features)
* **TF-IDF features at n-gram level**: adds a matrix of TF-IDF scores applied to ngrams in the sample. In this study, 1, 2, 3 grams were experimented and a max number of 10,000 features was applied (+10000 features).
* **Topic model as features**: adds a matrix of distribution probabilities over words in the sample. Latent Dirichlet Allocation (LDA) calculates the proportion of words in a sample that is currently assigned to the topic, and the proportion of assignments to the topic overall the entire data corpus (+40 features).
* **Different combinations of features:** creates a maximum feature matrix of 772 samples by 10,215 features.

## Convolutional Neural Network

This study created a Keras Sequential model that contained a stack of layers. These include the embedding layer as the input, a convolutional 1D layer, dropout layers, pooling layers, dense layers, and a fully connected layer as the output. Figure 4 shows a deeply connected sequential CNN model, with all added features, compiled using Keras with a total of 8,762,745 parameters. The details and parameters for each layer is as follows (in order):

* **Embedding layer as input**: In this study, the size of the vocabulary was approximately 1,515 words after pre-processing. The size of the dense vector – output dimensions – was experimented using 50, 100, 200, and 300. The maximum length of each sample sequence was 175; based on sample length distribution across the dataset (see Figure 1). The number of parameters to train in the layer is calculated by multiplying the vocabulary size and the dimension size.
* **Conv1D layer**:The function of this layer is to detect features using filters. The number of filters determine the number of features that is learned (or the number of words that should be considered at once). Using a maximum word length of 175, the output for the first neural network layer is a matrix of 175 by filter size. This study experimented with filter sizes 32, 64, and 128. The configuration of a Conv1D layer also calls for kernel size to determine the size of the window as it slides down each matrix (‘cut out’ by the filter). The study experimented with kernel sizes 3, 5, and 7.
* **Pooling layers**: The function of a pooling layer is to reduce model overfitting by down-sampling and reducing the size of incoming features. The CNN model in this study utilised both average pooling and global max pooling. In average pooling, the average value of all features in the pool is taken, and the rest are dropped. This study experimented with average pooling sizes of 1, 2, and 3. In global max pooling, the maximum of all features is taken, and the rest the vectors are dropped.
* **Dense layer:** The most important aspects of a connected neural network as each output node connected to a dense layer is a product an input node multiplied by a *weight* and a *bias*. The activation function Rectified Linear Unit (ReLU) is passed in this layer. In this study, the value of this dense layer was set to 10 - the dimensionality of the output (i.e. 1 by 10 neurons) - and indicates the reduction of dimensionality in preparation for the final output.
* **Dropout layer:** To further avoid overfitting, a drop layer was added to randomly assign 0 *weight* to neurones in the CNN model. In doing so, the model becomes less sensitive to small variations in the data, and more adaptable to new or unseen information. This study experimented with dropout levels of 0.25 and 0.5. The output of this layer is still 1 by 10 matrix of neurons as per previous layer.
* **Fully connected layer as the output**: The final layer reduces the 1 by 10 matrix of neurons from the previous layer to a vector with a height of 1. As a binary classifier, the output layer uses a Sigmoid activation function to determine whether a probability of seeing the label is closer to 0 or 1 – as per label values where 0 = NO, and 1 = YES.

|  |  |
| --- | --- |
| Layer (type) Output Shape Param #  =================================================================  embedding (Embedding) (None, 10215, 200) 303000  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv1d (Conv1D) (None, 10215, 128) 128128  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling1d (MaxPooling1 (None, 2553, 128) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv1d (Conv1D) (None, 2553, 128) 82048  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  max\_pooling1d (MaxPooling1 (None, 638, 128) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv1d (Conv1D) (None, 638, 128) 82048  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  flatten (Flatten) (None, 81664) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense (Dense) (None, 100) 8166500  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout (Dropout) (None, 100) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense (Dense) (None, 10) 1010  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout (Dropout) (None, 10) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense (Dense) (None, 1) 11  =================================================================  Total params: 8,762,745  Trainable params: 8,762,745  Non-trainable params: 0 | Figure 4 – Example of sequential model depicting a connected CNN including all features |

## Model Training

In this study, the sample dataset was split by 80% for training (772 samples) and 20% for testing (193 samples. To encourage maximum performance, the training and test sets were split for each level of LV (i.e. resampling with replacement). The models were trained using a 10-fold cross validation which partitions the training dataset into 10 folds; validating one-fold at a time across 10 iterations, while training the model on the other 9 folds with each iteration. This process aims to reduce a less biased model and remains separate from the testing set which remains completely unseen until model evaluation.

In this study, the number of epochs were experimented ranged from 10 to 100. In preliminary training runs, there is no significant improvements to accuracy and loss beyond 10 epochs. As such, the number of training epochs was fixed at 15 for the remainder of this study. The batch size was set to 64 which required 12 iterations to complete 1 training epoch.

Hyperparameter tuning was carried out using a parameter grid. The perimeter grid combines all possible permutations of hyperparameter values and trains the model using each combination. The parameters tested include the embedding dimensions (100, 200, 300) in the embedding layer, filter size (32, 64, 128) and kernel size (3, 5, 7) in the Conv1D layer; resulting the 27 combinations. After model training and testing, the grid reveals the best score and parameters used for model.

## Evaluation

The most important measure in this study was precision. Based on the off-the-shelf MLP results, the CNN model is expected to exceed a precision of 89% for *normal*, 39% for *mild*, 54% for *moderate*, and 55% of *severe*. Another notable measure is the F1-score which represents the weighted average of precision and recall. Unlike accuracy, F1-score focuses on the ‘outcome of interest’ and is more important to this study given its focus on minority classes (i.e. *moderate* or *severe*), and the costly risk to the patient if the minority class was undetected (i.e. false negatives).

# Results

Overall, the binary classifier performed well on binary classifications for each level of LV function. Each model was class-specific as the data was split, trained, and tested, for each individual class. This approach was done to obtain the maximum achievable precision for each class. In doing so, the maximum precision was 85.7% for *mild*, 74.1% for *moderate*, and 77.8% for *severe* (see Table 3). F1-score was unable to exceed the 80% threshold for any of the three minority classes. Surprisingly, ROC-AUC achieved 94 – 96% for the three minority classes, indicating excellent classification accuracy.

Table 3 - Performance of CNN as binary classifier for each level of LV function

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Class** | **Accuracy** | **Precision** | **Recall** | **F1-score** | **Kappa** | **ROC-AUC** |
| Conv1D Neural Network (CNN) | Normal | 0.968912 | **0.973913** | 0.973913 | **0.973913** | 0.93545 | 0.986845 |
| Mild | 0.917098 | **0.857143** | 0.731707 | **0.789474** | 0.738261 | 0.940629 |
| Moderate | 0.922280 | **0.740741** | 0.714286 | **0.727273** | 0.681973 | 0.968615 |
| Severe | 0.979275 | **0.777778** | 0.777778 | **0.777778** | 0.766908 | 0.963768 |

The tuned Conv1D model benchmarks well again the MLP and logistic regression models. However, this is an unfair comparison as the MLP and logistic regression models were used as multi-class classifiers, whereas the Conv1D model is a binary classifier and the results shown for each class were resampled with replacement.

Table 4 - Comparison – as binary classifier for minority class CNN vs MLP vs LogReg

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Performance** | **Conv1D CNN  (Tuned)** | **MLP  (Default)** | **Logistic Regression  (Default)** |
| Mild | Accuracy | 0.92 | 0.84 | 0.85 |
| **Precision** | **0.86** | **0.39** | **0.45** |
| Recall | 0.73 | 0.54 | 0.60 |
| F1-score | 0.79 | 0.46 | 0.52 |
| Moderate | Accuracy | 0.92 | 0.91 | 0.90 |
| **Precision** | **0.74** | **0.54** | **0.54** |
| Recall | 0.71 | 0.70 | 0.67 |
| F1-score | 0.73 | 0.61 | 0.60 |
| Severe | Accuracy | 0.98 | 0.96 | 0.97 |
| **Precision** | **0.78** | **0.55** | **0.73** |
| Recall | 0.78 | 0.75 | 0.73 |
| F1-score | 0.78 | 0.63 | 0.73 |

The accuracy and loss curves are showing in Figure 5 for *mild*, Figure 6 for *moderate*, and Figure 7 for *severe*. Notably, approximately 8 epochs are required before the loss function is minimised for both *mild* and *moderate*, and approximately 15 epochs for *severe*. At approximately 4 epochs (for both *mild* and *moderate* LVEF), the training / validation accuracy, and training / validation loss, begin to split and diverge. This indicates that training should be stopped or else a call-back function should be implemented shortly after 4 epochs for *mild* and *moderate*.

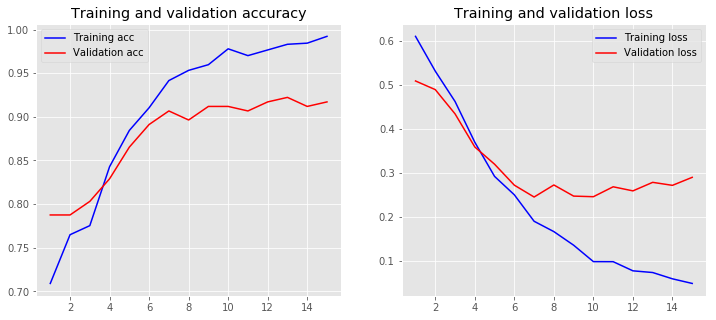


Figure 5 - Accuracy and loss curves (training versus testing) for mild LV function

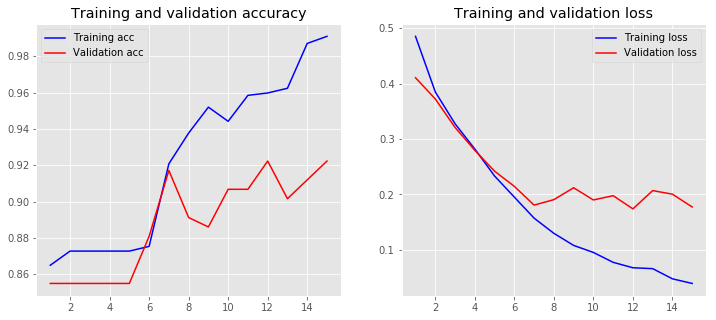


Figure 6 - Accuracy and loss curves (training versus testing) for moderate LV function

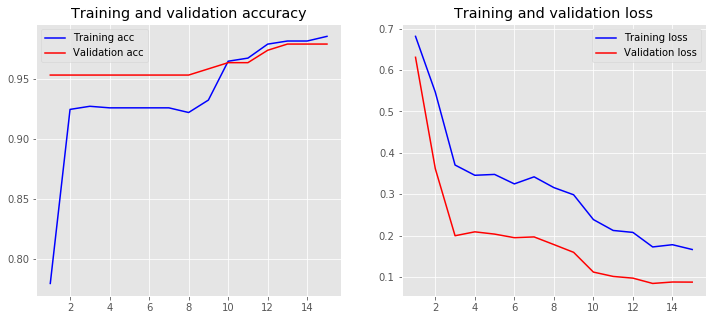


Figure 7 - Accuracy and loss curves (training versus testing) for severe LV function

# Discussion

Despite best efforts, the precision for detecting each of the three abnormal classes of LV function (*mild*, *moderate*, and *severe*) was disappointing. Clearly, the advantage of ‘resampling with replacement' could not induce a F1-score beyond 80% for any of the three classes. Furthermore, adding additional features to the embedding matrix such as a TF-IDF weighted matrix (either as *uni*-gram or *n*-gram) or an LDA topic distribution matrix, have little effect on improving precision or F-score beyond those reported in the results. This lack of performance improvement shows that -adding more features does not necessary equate to better precision. On the contrary, adding too many features could induce more noise into model training.

There were two mistakes made in pre-processing. The first was the failure to not remove negation *n*-grams which would have resulted in more noise. The second mistake was the removal of digits which exist to provide quantitative information that could be related to concept terms. In some cases, quantitative measures could be identified where qualitative measures could not. It is better to have fewer features with more defining characteristics about the training corpus than too many features that add noise.

The dataset in this study was affected by class imbalance – especially for *moderate* and *severe*. However, the literature review has eluded to the prospect that better precision (e.g. >85%) across each of the minority classes is obtainable with more appropriate pre-processing and feature engineering. For example, Xie et al. (10) focused their efforts on a classifier to primarily identify text descriptions and phrases relating to LV that would be used for making the LVEF assessment.

Since ECHO reports are not uniformly written, it was difficult to determine rules around how strings should be detected. In ECHO reports, there are four levels of variations: 1) Concept level diversity - e.g. ‘left ventricular ejection fraction’ could be written as EF, LVEF, left ventricular EF, or systolic left ventricular function, etc. 2) Diversity in how values are represented - e.g. the LVEF function could be reported as a qualitative measure such as “normal”, “reduced”, or “severe”, or be reported as a quantitative value such as 55%, 40%, or 35%. 3) Diversity in how values are linked to the concept – e.g. “Normal EF”, “EF = 55%”, “LVEF: 60%”. 4) Diversity in ECHO report template - to a lesser extent, the database storing ECHO reports vary between hospitals which result in differences between data structure, table shape, and naming convention. For example, data from Auckland City Hospital has an additional field called “LV Comments” whereas data from Middlemore Hospital do not.

For these reasons, Patterson et al. (9) emphasised the need for *concept identification*: one that uses a comprehensive lexicon to capture a wide range of terms associated with a target concept (e.g. “ejection fraction”). The dictionary should be bespoke because while standardised medical vocabularies such as the Unified Medical Language System (UMLS) would not necessarily capture locally defined abbreviations (e.g. “EF”), acronyms (e.g. “HF-rEF”), and spelling variations or misspellings.

## Next Step

The next step is to develop a text chunking algorithm to identify relevant concept, context, and measurement information. The chunking algorithm needs to capture phrases that contained keywords relating to LVEF and information relating to its assessment. The gold standard validation for this model would involve clinicians determining whether one phrase or multiple phrases are indeed used for the LV assessment. The advantages of a chunking algorithm can be seen in the example below – this is an assessment of *moderate* LVEF:

Mild LV dilatation. The basal LV segments and mid lateral walls are relatively active. All other regions are severely impaired with severe hypokinesis of the mid ventricular septum. The mid to apical anterior wall and all apical segments are akinetic. LV ejection fraction is moderately reduced. Calculated LV ejection fraction 35 % (apical BP MOD). Eccentric left ventricular hypertrophy. Moderate LV diastolic dysfunction with increased LV filling pressures (pseudonormalisation).

The phrases “*mild LV dilation*” and “*severely impaired*” are included in this sample. Despite these juxtapositions, the assessment for LV for this report was *moderate*. It was based on the phrases “*LV ejection fraction is moderately reduced*” and “*moderate LV diastolic dysfunction*”. This method of relating predefined keywords to concepts, contexts, and LV assessment information, was carried out as per NLP steps taken by Kim et al. (7) and Petterson et al. (9). The embedding matrix would have more defining *concept features, context features,* and less noise, which combine to ensure that concept-relevant phrases are meaningful.

# Conclusion

The true finding of this study was in discovering the need to develop a chunking algorithm that captures conceptual, contextual, and measurement features from ECHO reports. In doing so, there would be significant reduction in noise and significant improvement in engineering a feature set that is bespoke to medical text. While this study provided some useful benchmarks for a 1-dimensional CNN binary classifier, it was unable to exceed an F-score of 80% for *mild*, *moderate*, or *severe* LVEF, irrespective of how the training data was sampled, how deeply connected the neural network layers were, or well the model was tuned.

# References

(1) Little WC. Hypertension, heart failure, and ejection fraction Circulation; 118:2223 2008.

(2) Curtis JP, Sokol SI, Wang Y, Rathore SS, Ko DT, Jadbabaie F, et al. The association of left ventricular ejection fraction, mortality, and cause of death in stable outpatients with heart failure. J Am Coll Cardiol ;42:736-742 2003.

(3) Gaasch WH, Delorey DE, Kueffer FJ, Zile MR. Distribution of left ventricular ejection fraction in patients with ischemic and hypertensive heart disease and chronic heart failure. Am J Cardiol ;104:1413-1415 2009.

(4) Cardinale D, Colombo A, Bacchiani G, Tedeschi I, Meroni CA, Veglia F, et al. Early detection of anthracycline cardiotoxicity and improvement with heart failure therapy. Circulation ;131:1981-1988 2015.

(5) Wagholikar KB, Fischer CM, Goodson A, Herrick CD, Rees M, Toscano E, et al. Extraction of Ejection Fraction from Echocardiography Notes for Constructing a Cohort of Patients having Heart Failure with reduced Ejection Fraction (HFrEF). J Med Syst ;42:209 2018.

(6) Garvin JH, DuVall SL, South BR, Bray BE, Bolton D, Heavirland J, et al. Automated extraction of ejection fraction for quality measurement using regular expressions in Unstructured Information Management Architecture (UIMA) for heart failure. Journal of the American Medical Informatics Association ;19:859-866 2012.

(7) Kim Y, Garvin J, Goldstein MK, Meystre SM. Classification of Contextual Use of Left Ventricular Ejection Fraction Assessments. Stud Health Technol Inform ;216:599 2015.

(8) Kim Y, Garvin JH, Goldstein MK, Hwang TS, Redd A, Bolton D, et al. Extraction of left ventricular ejection fraction information from various types of clinical reports. *Journal of Biomedical Informatics.* 2017;67:42-48. http:<https://doi.org/10.1016/j.jbi.2017.01.017>.

(9) Patterson OV, Freiberg MS, Skanderson M, Fodeh S,J., Brandt CA, DuVall SL. Unlocking echocardiogram measurements for heart disease research through natural language processing. BMC Cardiovascular Disorders ;17:151 2017http:10.1186/s12872-017-0580-8.

(10) Xie F, Zheng C, Yuh-Jer Shen A, Chen W. Extracting and analyzing ejection fraction values from electronic echocardiography reports in a large health maintenance organization. Health informatics journal ;23:319-328 2017.

# Appendix

import os

import pickle

import numpy as np

from sklearn import decomposition

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer, CountVectorizer

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad\_sequences

import matplotlib.pyplot as plt

from keras.models import Sequential

from keras import layers

from sklearn.metrics import accuracy\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import cohen\_kappa\_score

from sklearn.metrics import roc\_auc\_score

# Data

os.chdir("D:/Research/Echo NLP Project/")

df = pickle.load(open("Research Report/Clean\_data\_v1.pkl", 'rb'))

df['severe'] = np.where(df['label']=="Severe", 1, 0)

df['moderate'] = np.where(df['label']=="Moderate", 1, 0)

# Model

def create\_model(num\_filters, kernel\_size, vocab\_size, embedding\_dim, input\_len):

model = Sequential()

model.add(layers.Embedding(vocab\_size, embedding\_dim, input\_length=input\_len))

model.add(layers.SpatialDropout1D(0.25))

model.add(layers.Conv1D(num\_filters, kernel\_size, padding='same', activation='relu'))

model.add(layers.AveragePooling1D())

model.add(layers.GlobalMaxPool1D())

model.add(layers.Dense(10, activation='relu'))

model.add(layers.Dropout(0.25))

model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

return model

# Main settings

epochs = 15

num\_words = 70000

batch\_size = 128

embedding\_dim = 200

num\_filters = 128

kernal\_size = 5

input\_len = X\_train.shape[1]

# Data / Label

sentences = df['fulltext']

y = df['moderate']

# Train-test split

sentences\_train, sentences\_test, y\_train, y\_test = train\_test\_split(

sentences, y, test\_size=0.25, random\_state=1000)

# TF-IDF - Word Level

tfidf\_vect = TfidfVectorizer(analyzer='word', token\_pattern=r'\w{1,}')

tfidf\_vect.fit(df['fulltext'])

X\_train\_tfidf = tfidf\_vect.transform(text\_train).toarray()

X\_test\_tfidf = tfidf\_vect.transform(text\_test).toarray()

# TF-IDF - ngram level

tfidf\_vect\_ngram = TfidfVectorizer(analyzer='word', token\_pattern=r'\w{1,}', ngram\_range=(1,2), max\_features=10000)

tfidf\_vect\_ngram.fit(df['fulltext'])

X\_train\_tfidf\_ng = tfidf\_vect\_ngram.transform(text\_train).toarray()

X\_test\_tfidf\_ng = tfidf\_vect\_ngram.transform(text\_test).toarray()

# Topic Feature using LDA

# Begins with BOW

count\_vect = CountVectorizer(analyzer='word', token\_pattern=r'\w{1,}')

count\_vect.fit(df['fulltext'])

# transform the training and validation data using count vectorizer object

X\_train\_count = count\_vect.transform(text\_train)

X\_test\_count = count\_vect.transform(text\_test)

# train a LDA Model

lda\_model = decomposition.LatentDirichletAllocation(n\_components=40, learning\_method='online', max\_iter=40)

X\_train\_lda = lda\_model.fit\_transform(X\_train\_count)

X\_test\_lda = lda\_model.fit\_transform(X\_test\_count)

# Word Embedding

# Use information from data exploration

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(text\_train)

X\_train\_emb = tokenizer.texts\_to\_sequences(text\_train)

X\_test\_emb = tokenizer.texts\_to\_sequences(text\_test)

vocab\_size = len(tokenizer.word\_index) + 1 # Adding 1 because of reserved 0 index

maxlen = 175 # Some samples go up to 200 words!

X\_train\_emb = pad\_sequences(X\_train\_emb, padding='post', maxlen=maxlen)

X\_test\_emb = pad\_sequences(X\_test\_emb, padding='post', maxlen=maxlen)

# Vanilla

X\_train = X\_train\_emb

X\_test = X\_test\_emb

# Combine TF-IDF (word level) with Word Embedding

X\_train = np.concatenate([X\_train\_tfidf, X\_train\_emb], axis=1)

X\_test = np.concatenate([X\_test\_tfidf, X\_test\_emb], axis=1)

# Combine TF-IDF (ngram level) with Word Embedding

X\_train = np.concatenate([X\_train\_tfidf\_ng, X\_train\_emb], axis=1)

X\_test = np.concatenate([X\_test\_tfidf\_ng, X\_test\_emb], axis=1)

# Combine Topic Feature LDA with Word Embedding

X\_train = np.concatenate([X\_train\_lda, X\_train\_emb], axis=1)

X\_test = np.concatenate([X\_test\_lda, X\_test\_emb], axis=1)

# Combine TF-IDF (word level) + Topic LDA + Word Embedding

X\_train = np.concatenate([X\_train\_tfidf, X\_train\_emb, X\_train\_lda], axis=1)

X\_test = np.concatenate([X\_test\_tfidf, X\_test\_emb, X\_test\_lda], axis=1)

# Combine TF-IDF (word level) + Topic LDA + Word Embedding

X\_train = np.concatenate([X\_train\_tfidf\_ng, X\_train\_emb, X\_train\_lda], axis=1)

X\_test = np.concatenate([X\_test\_tfidf\_ng, X\_test\_emb, X\_test\_lda], axis=1)

# Parameter grid for grid search

param\_grid = dict(num\_filters=[32, 64, 128],

kernel\_size=[3, 5, 7],

embedding\_dim=[100, 200, 300],

vocab\_size=[vocab\_size], # Fixed during vectorisation

maxlen=[maxlen]) # Fixed during vectorisation

model = KerasClassifier(build\_fn=create\_model,

epochs=epochs, batch\_size=batch\_size,

verbose=False)

grid = RandomizedSearchCV(estimator=model, param\_distributions=param\_grid,

cv=10, verbose=1, n\_iter=27)

grid\_result = grid.fit(X\_train, y\_train,)

# Evaluate testing set

test\_accuracy = grid.score(X\_test, y\_test)

print(grid\_result.best\_score\_)

print(grid\_result.best\_params\_)

print(test\_accuracy)