Medical Cost Prediction Tool documentation

-CS410: Text Information Systems

Fall 2019

Ishan Babbar (team lead) [ibabbar2@illinois.edu](mailto:ibabbar2@illinois.edu)

Houmin Zhong [houminz2@illinois.edu](mailto:houminz2@illinois.edu)

# Abstract

The Medical Cost Prediction Tool (MCPT) is designed to estimate the price of a doctor’s medical diagnosis and treatment if he/she has no insurance at all. The input of the MCPT is a short description of medical diagnosis. It can be sentence(s) or word phase(s), and the output is a cost range estimate.

# Introduction

Medical costs have confused and bewildered many Americans. In 2018, CNBC reported that health care costs are spiraling higher, but patient visits to a doctor have been on the decline. A growing number of consumers are staying away out of fear of big bills. However, “untimely visits or delay of visits to the physician ultimately leads to the increased cost of care,” the Cleveland Clinic’s CEO. Also, when people lose their jobs, they also lose their health insurance. If an unexpected illness or accident happens, knowing the cost of a procedure can help people to determine the acceptable price of treatment and estimate appropriate funds (Durand-Zaleski, 2008). The objective of the Medical Cost Prediction Tool (MCPT) is to estimate the price of a doctor’s medical diagnosis and treatment (if he/she has Medicare or no insurance at all).

# Related Work

Currently, popular websites like [FairHealthConsumer](https://www.fairhealthconsumer.org/) require patients to put in the exact procedure name or code to later give a cost estimate. This program is derived from the medical diagnosis/treatment and is a way to map the expenses. The MCPT tool simplifies this process and allows a direct mapping from your diagnosis to the expenses of your treatment.

# Methods

## Training Data description

Medicare National HCPCS Aggregate Summary Table data of 2015, 2016, and 2017 are obtained through <https://data.cms.gov/>. The descriptions of the column names and data types used in the MCPT model are listed in table 1. Only rows with facility data (value of 'F' in the Place of service column) was selected for modeling. Frist two rows of the dataset are shown in table 2.

**Table 1:** Data description of Medicare National HCPCS Aggregate Summary Table CY2017.

|  |  |  |
| --- | --- | --- |
| **Column name** | **Description** | **Type** |
| HCPCS description | Description of the specific medical service furnished by the provider. | Character |
| Place of service | Identifies whether the place of service submitted on the claims is a facility (value of 'F') or non-facility (value of 'O'). Non-facility is generally an office setting; however other entities are included in non-facility. | Character |
| Average submitted charge amount | Average of the charges that providers submit for the service. | Number |
| PriceRange | Average submitted charge amount is split into 6 different categories based on 0.1, 0.3, 0.5, 0.7, and 0.9 quantiles | Factor |

**Table 2:** First two rows of the dataset

|  |  |  |  |
| --- | --- | --- | --- |
| **HCPCS Description** | **Place of Service** | **Average Submitted Charge Amount** | **PriceRange** |
| Closed treatment of broken thumb with manipulation | F | 1708.37 | 4.ModerateHigh |
| incision and drainage of cavity behind eye | F | 2949.77 | 5.High |

## Data Preparation

Data cleaning steps, such as tokenization, stop word removal, and word stemming are performed. To reduce modeling difficulty, average submitted charge amount is split into 6 different categories based on 0.1, 0.3, 0.5, 0.7, and 0.9 quantiles, and the categories are shown in table 3. Data is split in to training, validation, and test sets in 60%, 15%, and 25%, respectively. In each of these train, validation, test datasets, we have ensured that they represent the same label distribution as seen in our data-prep.ipynb. This ensures that we’re training and validation on the same ground-truth since we don’t have an understanding for the samples of price-ranges we will expect.

**Table 3:** MCPT output medical treatment price ranges

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Price range | Very Low | Low | Moderate Low | Moderate High | High | Very High |
| High limit ($) | 121.9 | 660.3 | 1580.8 | 2647.1 | 4573.3 | 10269.4 |
| Low limit ($) | 23.8 | 121.9 | 660.3 | 1580.8 | 2647.1 | 4573.3 |

## Modeling

To be able to model this Text Classification task, we need to structure this medical problem in a way that complements our training and evaluation process. As we discussed earlier, we are going to have different thresholds to accommodate our price ranges so that we can use these labels for our classification needs. For more detailed work, please see the associated notebooks for each model in the file training.ipynb.

For our modelling purposes, we explored the following different architectures:

* Character-Level Convolutional Neural Network (CNN)
* Word-Level Recurrent Neural Network (RNN) – using LSTM Layer
* Word-Level RNN – using Bidirectional LSTM (BiLSTM) Layer

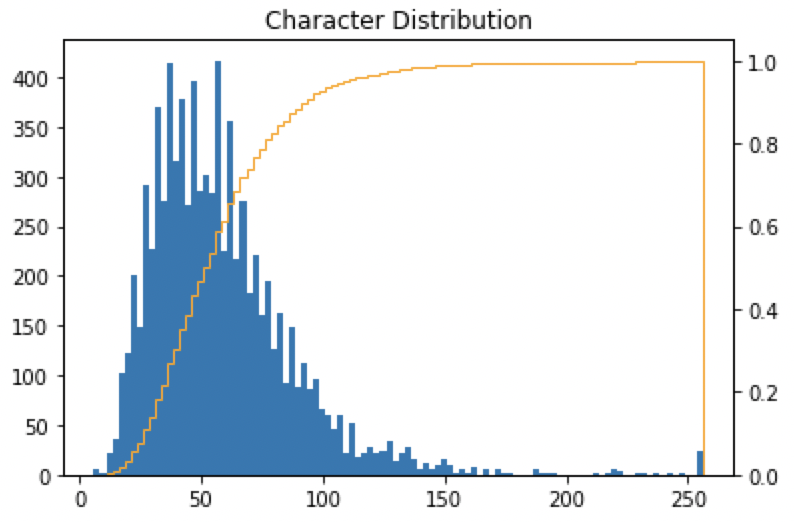
*CharCNN*

In our first model, we explored a character-level architecture that would utilize the subword information of all these medical descriptions. Since many of these descriptions have very few tokens, we thought that a convolutional-based model could use character-level information to help estimate these price ranges.

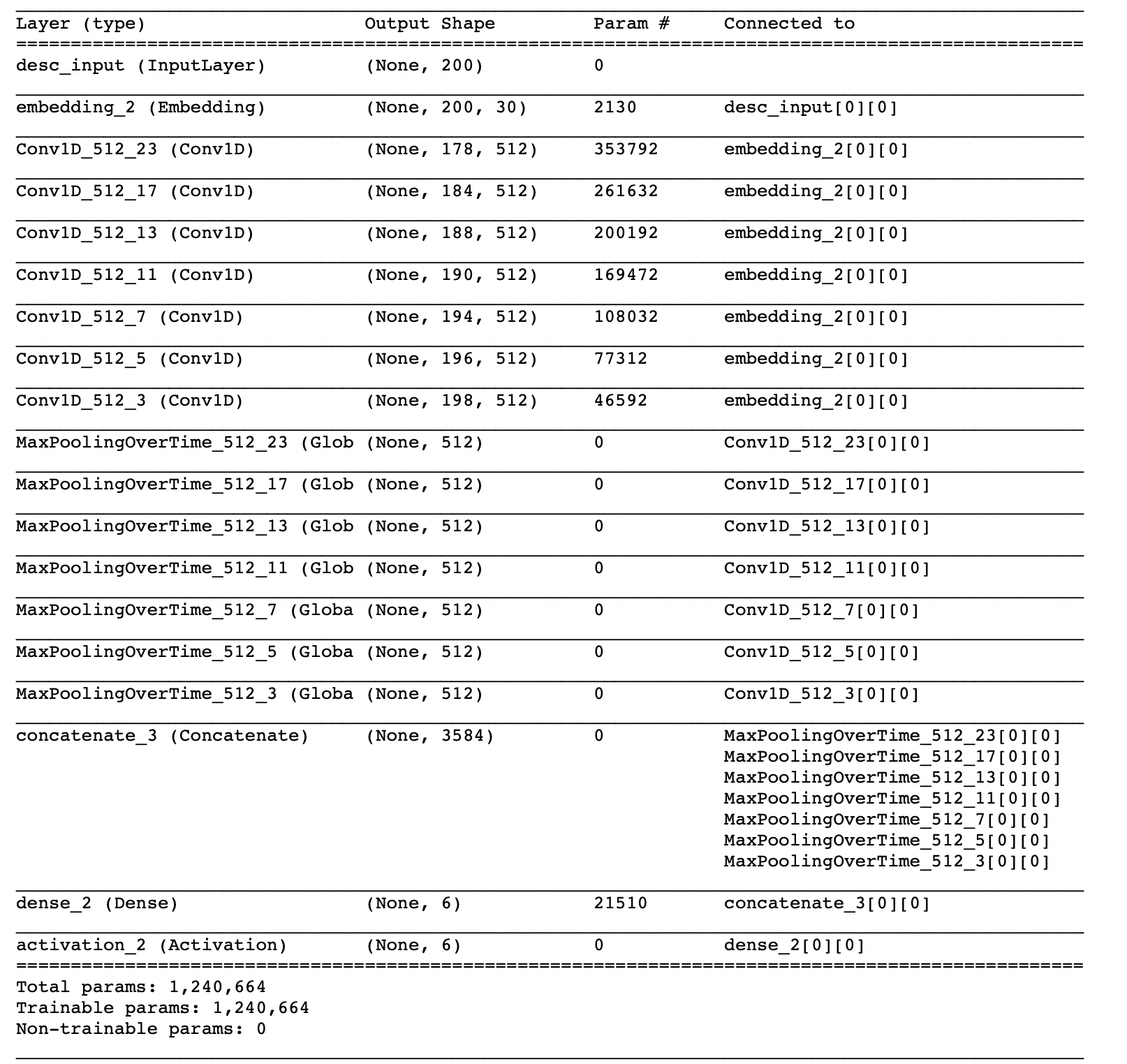
CNNs use multilayer perceptrons that have many applications in computer vision and natural language processing. Each convolution represents a window of words or characters and can be used to text classification problems. With various kernel sizes, we can utilize each word/character and detect strong patterns in the input.

As you can see, CNN work very well in creating dynamic features that then go through max-pooling layers which try to select the most relevant features in a particular region. This is extremely powerful in computer vision with 2D images, but the benefit has been applied to several 1D NLP problems. In our implementation, we’ll utilize several 1D convolutional layers with their associated max pooling steps to predict these word-level features.

Below is a distribution of the description characters, so that can help us understand how many features we’ll use to train our model with. As we can see there’s a much larger concentration of characters present in the first 150 characters and elongates to some descriptions having 250 characters.



This helped build our tokenizer which had a description length of 200 relevant character that would allow our text classification model to perform better. Since this is a text-based problem, we utilized a stack of 1-dimensional convolutional layers. This is how the architecture looks:



Intro to RNN:

Usually when humans are trying to understand text, we understand each based on the previous words. These sequential ordering is a natural way to understand text data. Traditional neural networks cannot use previous states to infer what is happening at every state of the sentence, for example.

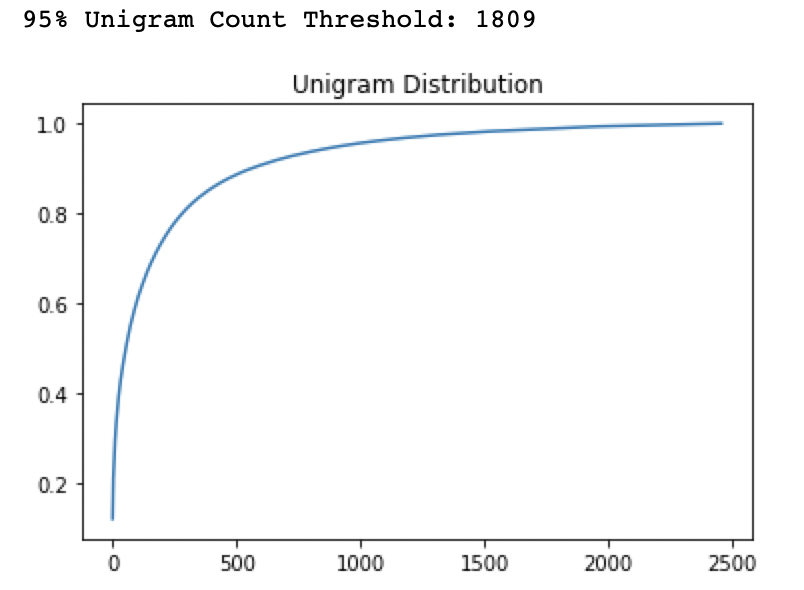
Recurrent Neural Networks (RNN) allows this chain of information to be passed through. It takes an input and then outputs a value representing the previous state that allow the next step in the loop to have this sequential and list-like structure. While RNNs can connect previous information to inform the understanding of the present frame, there are some scenarios that pose problems of long-term dependencies. In some scenarios, recent information might be weighted higher to perform the present task. And in other cases, more context is needed and there needs to be information accessible from previous states.

While RNNs can connect previous information to inform the understanding of the present frame, there are some scenarios that pose problems of long-term dependencies. In some scenarios, recent information might be weighted higher to perform the present task. And in other cases, more context is needed and there needs to be information accessible from previous states.

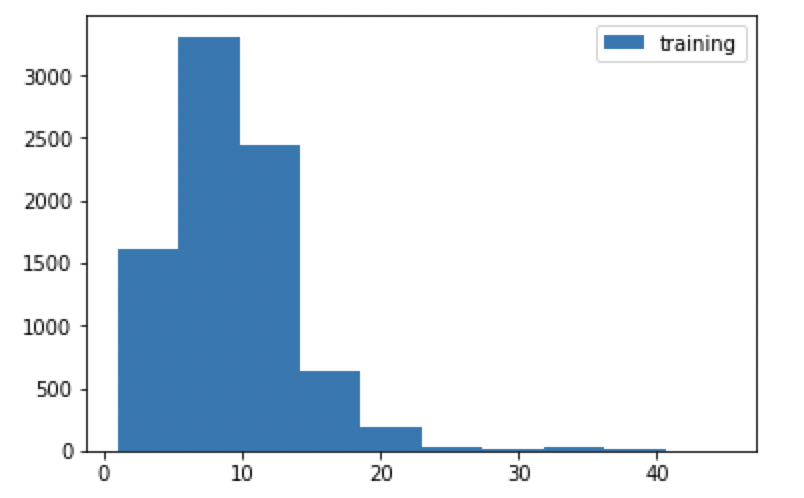
*LSTM*

In our second model, we used a unigram or word-level architecture to compare if these features were more indicative of the data than the character features. To begin, we first tried to identify the most common tokens present in our dataset. As you can see there’s a long tail of unique tokens, and we want to be able to measure a certain percentage of the most common tokens in our dataset.

We chose to use a 95% cut-off to measure the cumulative volume of the most common tokens, so that we can have a good representation of our dataset without having to have an overly large vocabulary. This is how that representation looks:

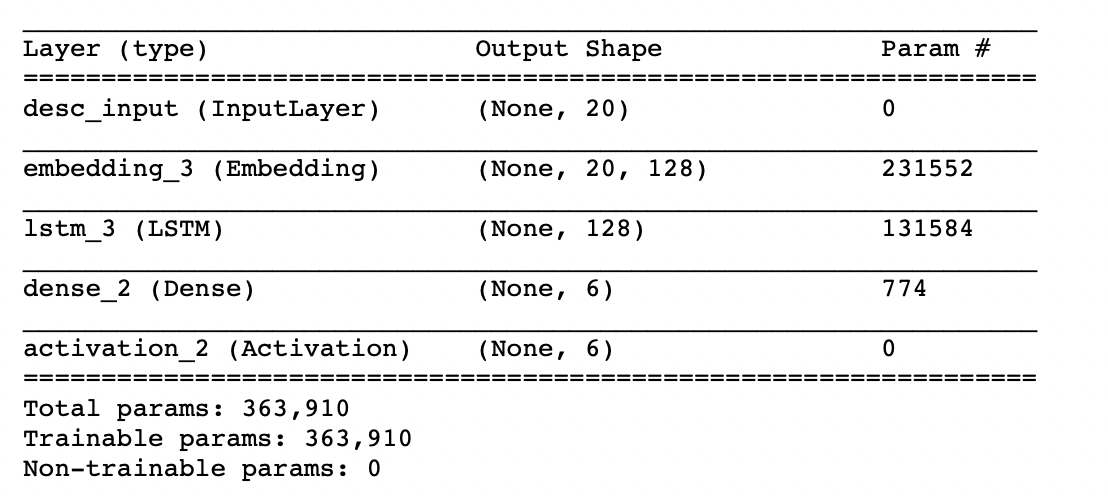


Aside from our vocabulary size, we also wanted to understand how long our embedding layer should be. This would be determined by investigating the average lengths of our training dataset’s descriptions. The following histogram shows that 20 tokens was a good cut-off point to represent our data with:



Additionally, with this unigram tokenizer we then will construct a new RNN architecture utilizing a Long-Short-Term-Memory (LSTM) layer. Long Short Term Memory (LSTM) networks were explicitly designed to combat the long-term dependency problem by explicitly removing or adding information to cell state. This type of RNN uses gates to use that sequential structure and utilize gates to control and protect each cell state. There are many layers and gate operations that go into defining an LSTM cell. What’s important here is to understand the structure in which data flows through this neural network. In our current problem, we can infer that this RNN structure works very well with tokens/words that can group them together and can provide valuable context. If we were to input at the character-level we would not see similar results, since independent characters might not be enough context to make meaningful predictions.

This architecture was not very deep and had significantly fewer parameters than the character-level CNN. Here’s how the architecture looked like:

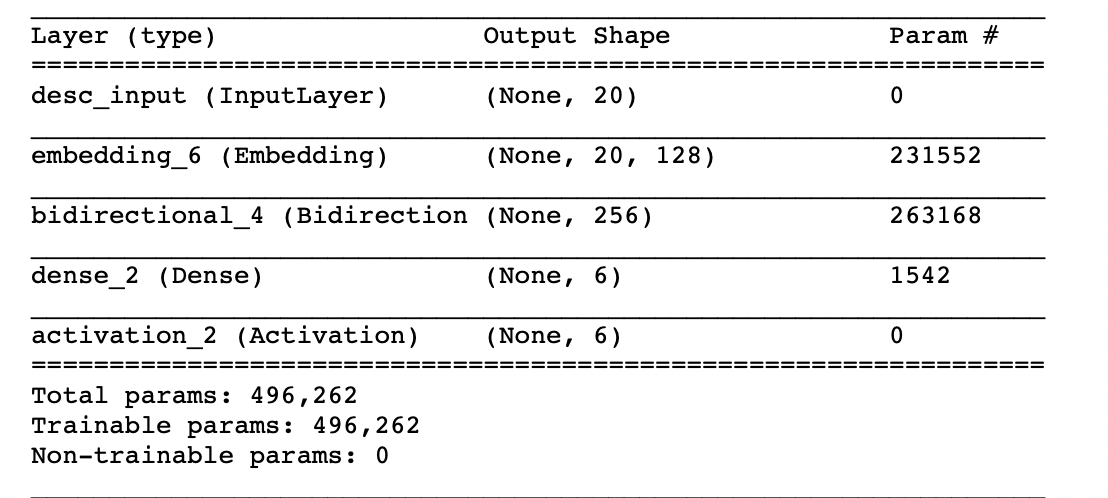


*BiLSTM*

For further investigation of these word-level architectures, we wanted to train our model on another RNN architecture – BiLSTM layer. The additional benefit is that the BiLSTM is essentially two LSTM concatenated together. While LSTMs perform very well, these cells only preserve information about the past or previous cell-states. This is why we also consider something called a bidirectional LSTM. This will allow us to run our inputs from past state to future state and from the future state to the past state. For example, in the sentence “hi my name is ishan”, if we’re at “name” this how both of these runs will look:

* Forward LSTM: “hi my”
* Backward LSTM: “is ishan”

By adding this layer of extra context, we can therefore utilise our word-level features to improve our model performance. This is fairly simple to implement, as we simply utilise a Bidirectional Layer on the LSTM in our previous model architecture. BiLSTMs are very commonly used in state-of-the-art text classification examples, so we applied it to try and analyze its performance with our classification task. The model’s architecture looked like the following:



As we can see this has more parameters than the LSTM architecture but less than the CharCNN architecture. We utilized the same description token length and embedding size as the LSTM for this model.

# Evaluation/Sample Results

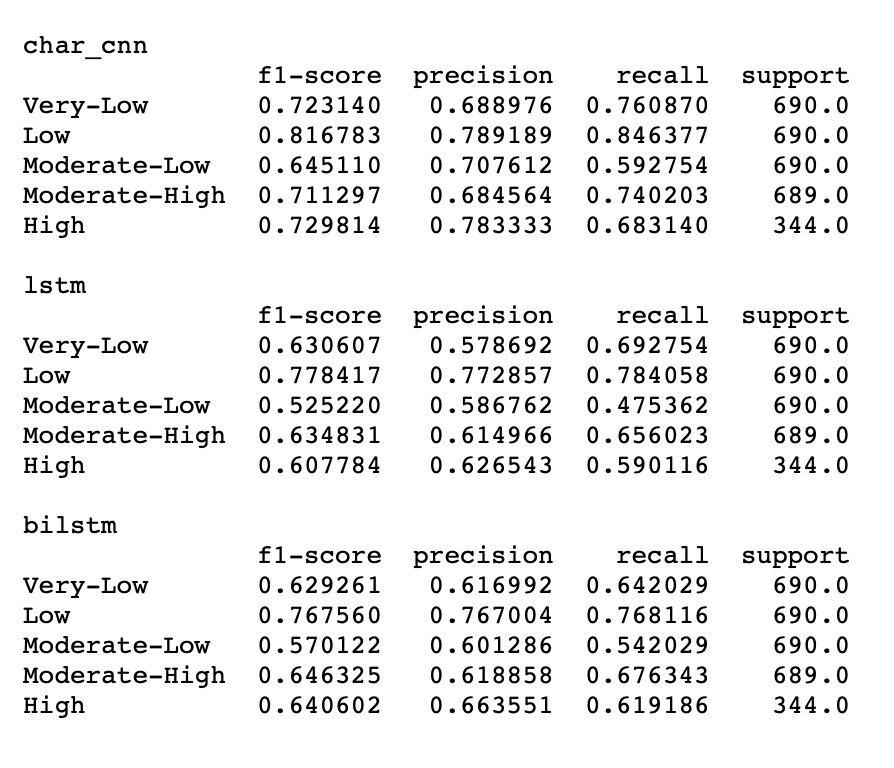
Input data is a short description of medical diagnosis, such as sentence(s) or word phase(s). The program output is price ranges and prediction confidence. Some example input and output are listed in table 4.

**Table 4:** example input and output

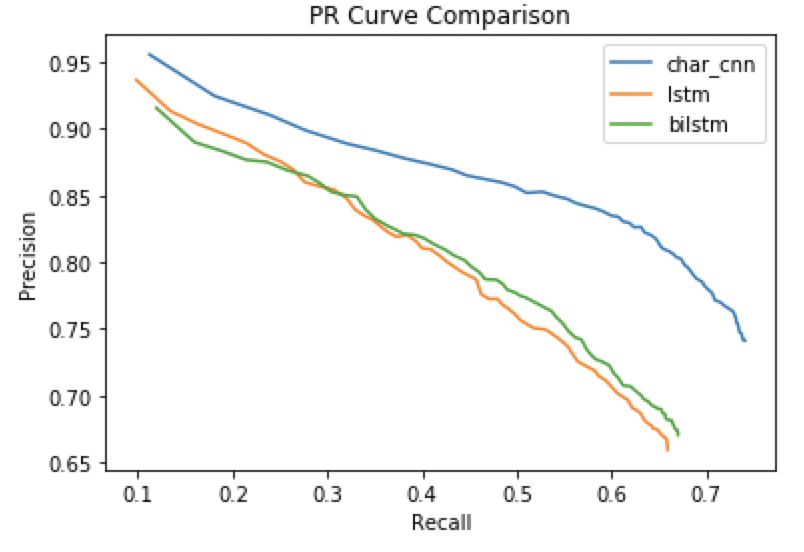
|  |  |  |
| --- | --- | --- |
| Input example phases | Output price range | Prediction confidence (%) |
| MRI scan brain | Low | 98.8 |
| Open treatment broken bone | High | 48.6 |
| Heart transplantation | Very-High | 40.2 |

Our evaluation results can be found in the GitHub in the comparison.ipynb notebook that’s based off our test.csv dataset. There are several ways to evaluate the performance of a model. We’ll be utilising precision and recall to construct curves and understand our thresholds for optimal performance. For simplicity, we’ll be evaluating each of these on a classification report, Precision-Recall (PR) and Recall Threshold curves.

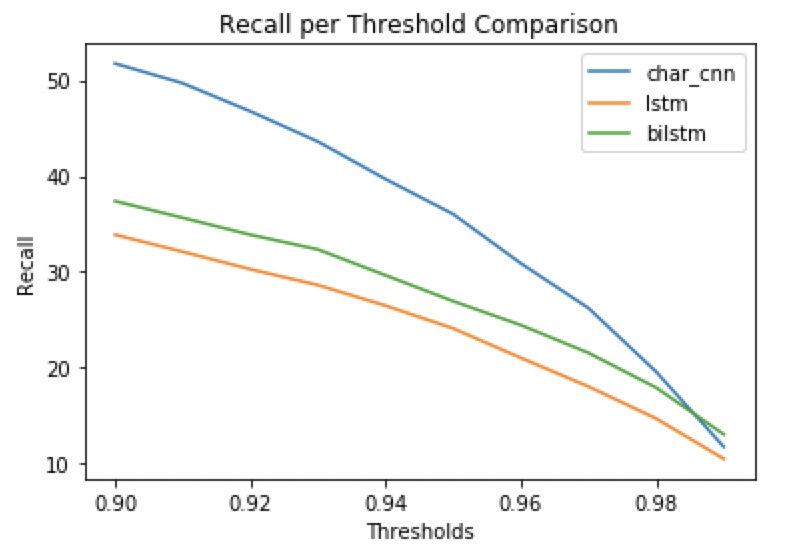
Our classification report allows us to look at the metrics (precision and recall + macro vs. micro) for each label. As we can see from our results, our CharCNN architecture performs better on all labels with our test data:



For more visually comparative procedures, we’ll look at how they compare across the PR and Recall/Threshold curves. The PR curve allows us to make the tradeoff between precision and recall metrics. When trying to understand a PR curve, we’re looking for the highest recall for a given precision and vice versa. As we can see the CharCNN outperforms both RNN word-level architecture once again:



For our recall comparison, we are trying to threshold for high recall. This curve allows us to make cut-offs for business applications to ensure that we have a high-take rate. If we wanted to threshold recall to ensure that our medical cost estimator was near perfect (~99.5% threshold), we would have to make a trade-off between the BiLSTM and CharCNN architecture:



## API setup (Windows 10)

1. Installing the requirements.txt - pip install –r requirements.txt
2. Run the Falcon API - python main.py
3. Run Postman (if you do not have it install, <https://www.getpostman.com>) to make inferences on the model
4. Create a request
5. Select “POST”
6. Enter “localhost:8080/invocations” in request URL
7. Enter text, such as “Heart Transplantation”
8. Click “Send”
9. Prediction should appear in Response section

Appendix: Individual Contribution

**Ishan Babbar:** Documentation, presentation, API building, data modeling + training, data cleaning

**Houmin Zhong:** Documentation, presentation, preliminary modeling, data cleaning, data finding

# Reference:

Durand-Zaleski, I. (2008). Why cost-of-illness studies are important and inform policy. *Vascular Medicine.* 13: 251–253.

Gillies, T. (2018). Why health care costs are making consumers more afraid of medical bills than an actual illness. *CNBC*.