Journal Title
XX(X):1–8
©The Author(s) 2025
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Predicting ICD9 Codes from Unstructured Clinical Notes Through the Use of NLP Techniques

Dimitrios Haralampopoulos and Izabel Miminoshvili

Abstract

By creating a tool that can identify ICD9 codes accurately and consistently from medical texts directly, we hope to reduce the burden on clinical coders and mitigate erroneous assignment of ICD9 codes. Previous studies either focused on identifying ICD9 codes for a specific patient or visit based on clinical notes [1,3] or implementing search-based methods [2]. We opted to use a variant Encoder-Decoder model, known as Text-to-Dual-Vectors, to predict ICD9 codes based on tokenized vectors of the input text and output two prediction vectors: one for diagnosis-related ICD9 codes and one for procedure-related ICD9 codes. We discuss what went well about our project, what didn't end up working, and what we would want to do in the future if we were to continue iterating on this model.

Keywords

ICD9 codes, NLP, BiLSTM, MIMIC-III, Health Informatics, Encoder-Decoder

Introduction

ICD9 codes are a staple in the world of medical billing and coding and require a lot of time and effort to be parsed from unstructured clinical notes and reports by clinical coders and accurately transcribed to ensure proper billing. There have been many attempts in the past to use Natural Language Processing techniques to ease the strain on clinical coders by creating assistive tools that automatically parse through the unstructured text and suggest possible ICD9 codes for the patient. In recent years, a growing body of work has explored the use of Natural Language Processing (NLP) for automating the assignment of ICD codes from clinical text, ranging from early statistical models to advanced deep learning systems.

However, this is no simple task, and there have been varying degrees of success of these tools. Coffman and Wharton (2009) [1] laid foundational work by experimenting with statistical learning methods to assign ICD-9 codes from radiology notes. Their findings emphasized the challenges posed by ambiguous language, coder disagreement, and the complexity of the ICD coding system, ultimately highlighting the limitations of purely statistical models in capturing medical context. Building on these early approaches, Zhao et al. (2023) [2] introduced a deep learning pipeline that leverages BERT-based embeddings and multi-label classification to predict ICD-10 codes from discharge summaries. Their method accounts for label imbalance and the hierarchical structure of the ICD system, significantly improving predictive performance over traditional baselines. This work reflects the increasing sophistication and real-world applicability of deep models in clinical coding tasks. Mullenbach et al. (2019) [3] further advanced the field by developing CAML, a convolutional attention-based model designed for interpretable ICD code prediction. Using the MIMIC-III dataset, their model outperformed standard CNNs and RNNs and introduced label-specific attention mechanisms that allowed clinicians to trace predictions back to relevant phrases in the source text. This combination of performance and explainability marks a critical step toward trustworthy AI in healthcare applications.

Fine-tuned pretrained language models (e.g. ClinicalBERT, BlueBERT) can extract diagnostic information from free-text notes with high accuracy [12]. Novel architectures that integrate code hierarchies or external knowledge (for example, the ICDXML system) have shown competitive performance on multi-label coding benchmarks [12]. A recent review observes that "recent advances in deep learning and natural language processing have been widely applied" to medical coding [9]. In practice, future coding systems will combine domain-specific fine-tuning with clever retrieval or hybrid methods to approach human-level coding performance.

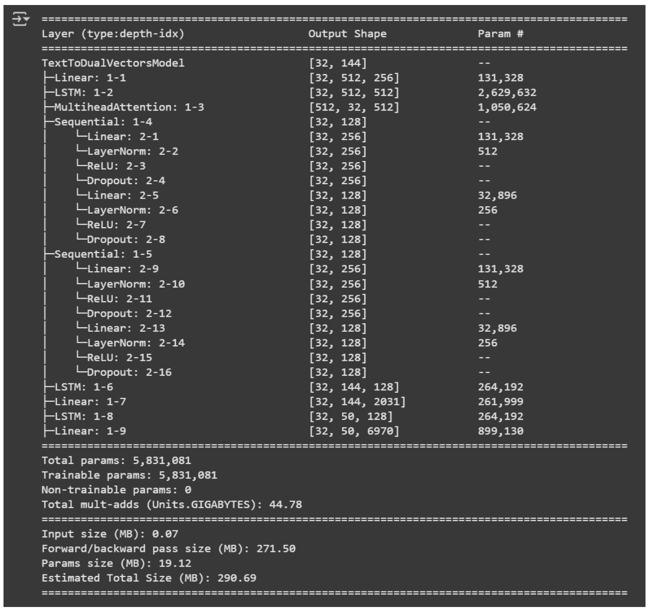
Automated coding faces significant data and modeling challenges. Clinical documents vary widely in format, length, and quality many contain redundant "note bloat" that can confuse NLP algorithms [11]. The ICD label space is vast and highly imbalanced: in ICU datasets like MIMIC-III, over half of ICD codes appeared in fewer than ten records, creating a "few-shot" learning problem [11]. In such long-tailed settings it is hard to learn robust models for rare diagnoses. Furthermore, deep learning systems often act as black boxes. As one study notes, even state-of-the-art automated coding systems "fall short of human performance" and "are 'black boxes,' not offering explanations" [10]. Ensuring model transparency and interpretability will thus be crucial to explain model predictions and align with clinical coding standards. We attempt to address at least some of these issues that have arisen in the domain of NLP for ICD9 prediction with this project. We aim to emulate systems like those presented in these past works and create something that at a baseline performs just as well if not better than benchmarks.

Data Processing

All data was pulled directly from PhysioNet. We used the MIMIC-III dataset, more specifically the NOTEEVENTS, DIAGNOSES_ICD, and PROCEDURES_ICD tables. For the system that we were doing the initial processing on, as well as on Google Colab free tier, the memory available to us was between 12 and 13 GB of RAM. This significantly impacted the way we chose to handle the data processing, as well as the construction of our model in the long run. The training datasets were created by first filtering NOTEEVENTS by note type of Reports and removing all rows labeled as errors in the ISERROR column. Afterwards, we cleaned the text in the TEXT column of NOTEEVENTS, replacing all whitespace with spaces. The text for each row was then tokenized using clinitokenizer, resulting in tokenized vectors of length 512, padded if necessary to ensure uniformity. In order to get all the ICD9 codes for each patient over all their reports, we joined on DIAGNOSES_ICD grouping by SUBJECT_ID and aggregating on ICD9_CODE and TEXT to create tensors of 512 length embeddings of varying sizes depending on how many records the patient had to their name. A separate instance of NOTEEVENTS was joined with PROCEDURES_ICD on SUBJECT_ID and TEXT, but aggregated only on ICD9_CODE, as TEXT would be redundant. The two joined tables were then joined with each other on SUBJECT_ID once again and separate columns were created for diagnosis and procedure ICD9 codes. The TEXT column was then transformed such that each tensor was averaged internally to create a single average token embedding for each subject for all their texts. This was done because it was difficult to automatically determine temporal sequencing for each document and what ICD9 codes were assigned to each note, thus it made sense to map all of the ICD9 codes for the patient to the concatenation of all their records and try and learn on those instead of each record individually. Vocabularies for diagnosis and procedure ICD9 tokens were created as they were being mapped to the dataframe. This yielded the final schema of {SUBJECT_ID: INT64, TEXT: LIST[INT], ICD9_DIAG: LIST[INT], ICD9_PROC: LIST[INT]}. These list values were then stored as string literals when written to the csv file that would become the training set. This data processing took roughly an hour and a half on native hardware, with optimizations made for memory management and the use of lazy evaluation of Polars dataframes, and was reduced to a size of 343MB whereas the original text data has an uncompressed size of 3.73GB. There is the potential that multiple records exist for the same subject due to the chunking process used while doing the data processing and writing to the file. This likely doesn't have much bearing on the results but in an ideal scenario where memory is not an issue, there would be no need to chunk the dataset during process and therefore no opportunity to have multiple tokenized embeddings for the same subject. Our train/test/validation split for the data was 80/10/10.

Methods

Once the data processing was completed and we obtained the necessary data that would be used for the model, we used the data to train a Text-to-Dual-Vectors model. We chose to use this model because it is able to take text input and output two vectors that allows us to see the predictions in the form of tokens. The two vectors shown as outputs are the diagnosis ICD9 codes and the procedures ICD9 codes. Furthermore, we used a Bidirectional Long Short-Term Memory (BiLSTM) network to allow information to flow from both forward and backward [4]. The BiLSTM uses two separate LSTM layers, forward LSTM and backward LSTM, and outputs the combination of these two layers as the result [4]. This enables the capture of more information to help with training of the model [4]. The structure of the model was slightly modified compared to a conventional Encoder-Decoder model in that it outputs two vectors, one for diagnosis ICD9 code predictions and one for procedure ICD9 code predictions. This required us to create our own loss function, which we called Dual Sequence Loss. This was, in effect, a wrapper for two instances of Cross Entropy Loss (one for each output vector) and was subsequently averaged before backpropagation. This was to ensure that the model would equally weight the output vectors during training. During a forward pass through the model, the input, starting with shape (BATCH_SIZE, 1, 512), was squeezed to eliminate dimensions of 1. A padding mask was then created for the tensor of (BATCH_SIZE, 512). The input is then sent through a linear projection layer to create a (BATCH_SIZE, 512, 256) tensor and then sent through a 2-layer encoder BiLSTM with a small dropout rate of 0.1. After this, a key padding mask was created for the Multi-Head Attention layer, which the encoder outputs were sent through along with the padding mask. After getting the attended outputs, we reshape the padding mask and get a ”global representation” of our input, which is in effect average pooling of the outputs. We then feed these global representations to the different output heads, which are sequential networks built from a Linear layer which is then normalized, ReLU activated, regularized with Dropout, and then repeated once more. This brings our output shape from each head to (BATCH_SIZE, 128). The head features are then unsqueezed and repeated to create the sequence inputs and are then fed to generators for the outputs which are also 2-layer BiLSTMs with Dropout. This gives us an output sequence of (BATCH_SIZE, 144, 128) for diagnoses and (BATCH_SIZE, 50, 128) for procedures. These were then sent through linear projection layers such that the last dimension of the resulting tensor matched the size of the vocabularies for diagnosis and procedure ICD9 codes, yielding final output shapes of (BATCH_SIZE, 144, 2031) and (BATCH_SIZE, 50, 6970) respectively. Argmaxed outputs and logits for both output tensors are returned by the model and used during each pass. Due to Google Colab usage limits, we were only able to train four models: Two (10 and 20 epochs each) models that ignored padding tokens in the loss function, and two models that took padding tokens into account in the loss function. This was done to see if the model would learn better from having a more enforced order of the output tokens given padding. Batch sizes for all models trained were set to 32. The 10 epoch models on the standard Colab free-tier T4 GPU took an hour and a half to train, and the 20 epoch models were trained with Colab's paid A100 GPU instance with High-RAM, which took just over an hour to train each.



```
===========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
===========================================================================================
TextToDualVectorsModel                   [32, 144]                 --
├─Linear: 1-1                            [32, 512, 256]            131,328
├─LSTM: 1-2                              [32, 512, 512]            2,629,632
├─MultiheadAttention: 1-3                [512, 32, 512]            1,050,624
├─Sequential: 1-4                        [32, 128]                 --
│    └─Linear: 2-1                       [32, 256]                 131,328
│    └─LayerNorm: 2-2                    [32, 256]                 512
│    └─ReLU: 2-3                         [32, 256]                 --
│    └─Dropout: 2-4                      [32, 256]                 --
│    └─Linear: 2-5                       [32, 128]                 32,896
│    └─LayerNorm: 2-6                    [32, 128]                 256
│    └─ReLU: 2-7                         [32, 128]                 --
│    └─Dropout: 2-8                      [32, 128]                 --
├─Sequential: 1-5                        [32, 128]                 --
│    └─Linear: 2-9                       [32, 256]                 131,328
│    └─LayerNorm: 2-10                   [32, 256]                 512
│    └─ReLU: 2-11                        [32, 256]                 --
│    └─Dropout: 2-12                     [32, 256]                 --
│    └─Linear: 2-13                      [32, 128]                 32,896
│    └─LayerNorm: 2-14                   [32, 128]                 256
│    └─ReLU: 2-15                        [32, 128]                 --
│    └─Dropout: 2-16                     [32, 128]                 --
├─LSTM: 1-6                              [32, 144, 128]            264,192
├─Linear: 1-7                            [32, 144, 2031]           261,999
├─LSTM: 1-8                              [32, 50, 128]             264,192
├─Linear: 1-9                            [32, 50, 6970]            899,130
===========================================================================================
Total params: 5,831,081
Trainable params: 5,831,081
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 44.78
===========================================================================================
Input size (MB): 0.07
Forward/backward pass size (MB): 271.50
Params size (MB): 19.12
Estimated Total Size (MB): 290.69
===========================================================================================
```

Figure 1. Model structure with parameters according to torchinfo

Results

Evaluating the effectiveness of the model is less straightforward than one would imagine. For each model, we tested Accuracy in three different ways, and Recall, Precision, and F1 Score in two ways. We did not record loss values for the models, mostly because the decrease in loss per epoch was incredibly small, such that it can hardly be considered significant. Something to note is that the loss for the ignore models was substantially higher than the models where padding tokens were not ignored. Loss for the ignore models floated around 4.0 or 5.0, while for the not ignoring models it remained around 0.6 to 0.7. This is likely due to the number of padding tokens the model was getting correct in the not ignoring models, and tells us very little about how well the model actually learned on the data. The three ways we computed metrics were as follows:

1. Creating a multilabel confusion matrix and iterating through each class label and averaging metrics from there.
2. For each set of predictions and ground truths, get the set of unique ICD9 codes present, discarding the padding token index from each. For each item in the prediction set, if it exists in the ground truth set, we add 1 to correct classifications. If not, we add 1 to misclassifications. These are then averaged over the total number of predictions made by the model to get ratios of correctly classified to misclassified codes.
3. Using sklearn's accuracy, recall, precision, and F1 score functions computing over flattened lists of labels and predictions, with the recall, precision, and F1 score having weighted averages and accounting for zero-divisions when relevant.

Our metrics for each model can be neatly summarized by these tables:

	Ignore (10ep)	Ignore (20ep)
Acc	0.9996	0.9996
Rec	0.0838	0.3319
Prc	0.0004	0.0440
F1s	0.0008	0.7318
	Not Ignore (10ep)	Not Ignore (20ep)
Acc	0.9999	0.9999
Rec	0.4997	0.4997
Prc	0.2503	0.2503
F1s	0.2989	0.2989

Table 1. Class Confusion Matrix Metrics (:.4f)

	Ignore (10ep)	Ignore (20ep)
Cor	0.0031	0.1548
Mis	0.9969	0.8452
	Not Ignore (10ep)	Not Ignore (20ep)
Cor	0.1207	0.1207
Mis	0.8793	0.8793

Table 2. Correct vs Misclassifications over total predictions (:.4f)

	Ignore (10ep)	Ignore (20ep)
Acc	2.2229e-5	0.0042
Rec	2.2229e-5	0.0042
Prc	8.4772e-7	0.0007
F1s	1.6214e-6	0.0012
	Not Ignore (10ep)	Not Ignore (20ep)
Acc	0.9022	0.9022
Rec	0.9022	0.9022
Prc	0.8335	0.8335
F1s	0.8662	0.8662

Table 3. Average over flattened predictions (:.4f)

As evident in the tables, the metrics reported are all over the place. It was clear to us that the accuracy rates provided by the class confusion matrix were greatly exaggerated compared to the reality of the situation, and thus we wanted to evaluate our model as best we could by trying different evaluation methods for metrics. All metrics between the Not Ignoring models are exactly the same, and was confirmed to be so as they were both retrained and evaluated again. Increasing the number of epochs seemed to have no bearing on the accuracy of the Not Ignoring models. The Ignoring models, however, started off very poorly with the 10 epoch version, with it correctly predicting tokens only 0.0031% of the time. After 10 more epochs, it improved dramatically, going to a 15% correct prediction rate. The accuracies for class predictions didn't change for the Ignoring models between 10 and 20 epochs, but every other metric increased significantly. Averaging over the flattened predictions, however, the Ignoring models performed very poorly compared to the Not Ignoring models. This, however, can be attributed to the amount of correctly predicted pad tokens in the outputs, since these metrics did not ignore padding tokens in their calculations while the others did. Overall, the best metric that we found for evaluating this model was Correct vs Misclassifications over Total Predictions. This metric gave us the insight we were looking for into how often the model correctly recommends relevant ICD9 codes regardless of sequencing. With this in mind, it appears that the Ignore model run for 20 epochs produces the highest number of correct token predictions overall compared to the other three models. This is somewhat promising, as it is an indication that the accuracy may be able to improve further if trained for more epochs.

Discussion

The results obtained from our work on this project highlight a number of valuable lessons regarding predicting ICD9 codes from unstructured medical text. Firstly, it's a difficult process. Compared to other model types, parsing natural language data into a machine-digestible format to be learned on is time-consuming and requires a lot of effort and planning. Without such structure, mistakes can pile up and even when the data collection is "complete", the data might not suit the model and vice versa. A lot of compromises were made on our end to create the dataset such that we would have ample time to process and train it. Ideally, we would have liked to include all types of notes and perhaps even temporally assign the ICD9 codes to each note for each subject. Our computational and chronological restraints hindered our ability to create the ideal version of the dataset, and could have impacted our results.

It should also be known that the pretrained tokenizer we used for medical text, clinitokenizer, is somewhat of a black box. We used it because it was freely available and showed promising results for tokenizing medical jargon, which is generally very difficult. Something that we were unaware of was the fact that the model, to our knowledge, would only yield the token ids of the text rather than embedding vectors. This means that we were learning on the token ids rather than the embeddings for each word, which could have added more depth to the model and the patterns that it could find, potentially leading to better results.

Using an encoder-decoder model was a solid choice for this task, but it may have not been the most optimal one. Encoder-decoder models have problems such as information bottlenecking via vanishing gradient, their computational complexity, inability to capture longer-term dependencies especially if RNNs are used, and their tendency to overfit as model size scales or if the dataset is too small. There are other models that are certainly worth testing out for ICD9 code prediction tasks such as search-based methods used in [2], transformer models, and perhaps even tree-based models. Given more time, we would have liked to further explore these different types of models and see how they perform for this task.

Reflection

Reflecting on our work for this study, there are some things that we did not get to accomplish and some things we needed to changed to accommodate for limitations. Originally, we planned to use NLP systems such as Encoder-decoder models or Transformers. However, we ended up using an Encoder-decoder variant called Text-to-Dual-Vectors. The main reason for this is because we were concerned about Transformers ability to accurately identify the ICD9 codes. A Transformer model requires computational power since the number of parameters needed grows quickly [5]. We did not have the computational power to handle training a Transformer model.

Furthermore, we did not use BLEU score as a metric to test the model because we did not evaluate text outputs. Thus, it was not necessary to use it since we tokenized the input data. To replace BLEU score, we used the following metrics: accuracy, recall, precision and F1 score. We decided to include recall and precision since we are dealing with a classification problem.

Although we tried our best to process the data before training the model, the data itself was not in the best condition for us to use. There was a lot of missing values and we had to create a separate dataset file from the MIMIC-III dataset because we did not all of the dataset files provided. We had to optimize the data to suit our model.

While training the model we noticed that the model construction bears a heavy weight on results, which meant that we had to constantly alter the model with different amounts of layers, dropouts, etc to see what combination would give us the best results. However, this is a very time-consuming process because we had to constantly reconstruct the model and train it. Unfortunately, time was a constraint for us because we did not give ourselves enough time to properly train the

model and improve it.
We also realized that we are dealing with a problem that has no perfect solution, so cannot expect for our model to work immediately. This is a problem that many researchers are still working out a solution for.

Division of Labor

Since each group consisted of two to three, our group of two tried to split the work to be fifty-fifty. However, it turned more into sixty-forty; sixty for Dimitri Haralampopoulos and forty for Izabel Miminoshvili. The main reason for this is because Haralampopoulos had more knowledge on constructing the model; thus, the task of building and training the model was handled by Haralampopoulos. He also helped with the presentation and the writing of the paper. The other member Miminoshvili helped with creating a shared folder to organize the presentation slides and necessary documents, writing the paper and training the model.

Limitations

While we were able to acquire results, there were a few challenges and limitations faced when it came to processing the data and training the model. One such challenge was computing power. We used Google Colab to run our model mainly for two reasons. The first reason was the fact that it was free to use as long as the user has a gmail account. The second reason was that we were able to run the model on GPU rather than a local CPU. This allowed us to save computational power and RAM on our local device. However, Google Colab has a time limit of about an hour to two hours for each user runtime. Thus, this limited the number of hours we could spend training the model. Training the model took the longest time to complete even with GPU. As a result, we were only able to run a low number of epochs. To deal with the user runtime limit, one member paid for time, but it did not help to improve the time it took to train the model.
Another challenge we ran into was tokenization. There are few public tokenizers available for clinical text data. As a result, it was difficult to find a good tokenizer that could help us tokenize our data. We needed to tokenize our data since we are dealing with text and numerical data. It made comparison of results easier to tokenize the text data. Although we were able to find a tokenizer, it is not perfectly tailored to our study. This is an issue because the tokenization could have impacted the training of the model resulting in skewed outputs. A concern we had with using a public tokenizer was that the ICD9 vocabularies built from the tokenizer might not be able to capture the structure of the codes as well as we would like, which would also effect the results of the training.
The biggest limitation we faced was the volume of the MIMIC-III dataset provided. It took us a long time to compress the data we needed for the model from 3.73GB to 343MB. Despite compressing the data, it still took a lot of computational power to process the data, which allowed for more mistakes and limitations in what we could when it came to processing. Overall, we were limited in the computational power provided to us and overwhelmed by the volume of the data.
It seems that in the end, time management was our greatest enemy. From the time that we had written the project proposal until the time that we had to submit this paper, we had about a month's worth of time to start working on the project. Due to work for other courses, the happenings of life, and our own hubris, we foolishly constrained ourselves to just over a week to finish this project. While we were able to produce results, they may have been much more promising if we had spent even double the time working on this project than what we had.

Future Applications

In the future, it would be beneficial for researchers to rely on our results to determine the extent to which NLP systems are reliable when identifying ICD9 codes based on unstructured MIMIC-III data. With no current solution to the ongoing struggle of ICD9 assignment for clinical coders, we believe that researchers would be able to clearly understand the limitations and advantages of using NLP systems such as the Text-to-Dual-Vectors model for identifying ICD9 codes with unstructured data based on our model.
Our study can serve as a foundation to expanding the model to train ICD10 or ICD11 code systems to see how well the model can perform with more current systems. These systems introduce more complexity than the ICD9 codes providing the

opportunity to test the scalability and adaptability of our model. The same methodology of first tokenizing the clinical notes before training the model that we used for the ICD9 codes can be used on the ICD10 or ICD11 codes. However, researchers should prepare to face the same challenges that we faced when training the model such as computational power and the size of the data provided. Despite these challenges, we believe that a Transformer model would make this possible since they are able to handle large and complex datasets [13].

Furthermore, our approach can be used to test the performance of the model on clinical notes in other languages or multilingual datasets. It would be a great way to broaden However, this might be difficult and would require significantly more resources to accomplish because the notes would need to be translated first to one language and then tokenized to be used on the model.

With the gradual transition from ICD9 to 10, and soon ICD-11, interoperability of these systems with the new standards is paramount. Future NLP systems for medical coding will need strategies for code mapping or transfer learning to bridge old and new standards. Further research will be needed to develop training corpora and algorithms that can handle multiple coding versions and "post-coordination" rules inherent in ICD-11.

AI governance and ethics will also have strong influence on the future of NLP-based medical coding. In many jurisdictions, AI-based medical coding tools will be treated as high-risk systems requiring strict oversight. For instance, the EU's forthcoming AI Act classifies AI in medical devices as "high risk," mandating comprehensive risk management, data governance, transparency, and human oversight [11]. U.S. regulators likewise stress transparency: FDA guidance on machine-learning software emphasizes that stakeholders must understand a model's "logic" to detect errors and biases [7]. For widespread adoption there would not only need to be a streamlined interface for the model but also a way to ensure the privacy and interpretability of the data and the explainability of the model. Beyond the model having high accuracy, it would also need to be compliant with government regulations such as HIPAA and with other guidelines set by the FDA, NIH, and other government organizations. Ethically, developers must audit for bias and fairness (ensuring, for example, that underserved patient groups are not systematically miscoded) and provide mechanisms for audit and correction. Technical advances in NLP-based medical coding will need to be paired with explainable models and robust governance structures to ensure safe, equitable adoption in healthcare in the future [7, 10].

References

[1] Coffman, Abe, and Nat Wharton. Clinical Natural Language Processing Auto-Assigning ICD-9 Codes. https://courses.ischool.berkeley.edu/i256/f09/Final%20Projects%20write-ups/coffman_wharton_project_final.pdf

[2] Falter, Maarten, et al. "Using Natural Language Processing for Automated Classification of Disease and to Identify Misclassified ICD Codes in Cardiac Disease." European Heart Journal. Digital Health, 9 Feb. 2024, https://doi.org/10.1093/ehjdh/ztae008.Accessed20Apr.2024.https://academic.oup.com/ehjdh/article/5/3/229/7604129

[3] Gangavarapu, Tushaar, et al. "Predicting ICD-9 Code Groups with Fuzzy Similarity Based Supervised Multi-Label Classification of Unstructured Clinical Nursing Notes." Knowledge-Based Systems, vol. 190, Feb. 2020, p. 105321,https://doi.org/10.1016/j.knosys.2019.105321.Accessed6Oct.2021.https://www.sciencedirect.com/science/article/abs/pii/S0950705119305982?via%3Dihub

[4] GeeksforGeeks. (2023, June 8). Bidirectional LSTM in NLP. GeeksforGeeks.https://www.geeksforgeeks.org/bidirectional-lstm-in-nlp/#

[5] Tom, B. (n.d.). Retrieved April 28, 2025, from https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1204/reports/custom/report21.pdf

[6] Aboy, M., Minssen, T., & Vayena, E. (2024). Navigating the EU AI Act: implications for regulated digital medical products. npj Digital Medicine, 7, 237. https://doi.org/10.1038/s41746-024-01232-3

[7] Food and Drug Administration. (2023).

Transparency for machine learning–enabled medical devices: Guiding principles. U.S. Department of Health and Human Services. Retrieved from https://www.fda.gov/media/164139/download
[8] Falter, M., Godderis, D., Scherrenberg, M., Kizilkilic, S. E., Xu, L., Mertens, M., et al. (2024). Using natural language processing for automated classification of disease and to identify misclassified ICD codes in cardiac disease. European Heart Journal – Digital Health, 5(3), 229–234. https://doi.org/10.1093/ehjdh/ztae008
[9] Ji, S., Sun, W., Li, X., Dong, H., Taalas, A., Zhang, Y., et al. (2024). A unified review of deep learning for automated medical coding. ACM Computing Surveys. Advance online publication. https://doi.org/10.1145/3664615
[10] Puts, S., Zegers, C. M. L., Dekker, A., & Bermejo, I. (2025). Developing an ICD-10 coding assistant: Pilot study using RoBERTa and GPT-4 for term extraction and description-based code selection. JMIR Formative Research, 9, e60095. https://doi.org/10.2196/60095
[11] Venkatesh, K. P., Raza, M. M., & Kvedar, J. C. (2023). Automating the overburdened clinical coding system: Challenges and next steps. npj Digital Medicine, 6, 16. https://doi.org/10.1038/s41746-023-00768-0
[12] Wang, Z., Wang, Y., Zhang, H., Zhang, X., He, Y., & Wu, H. (2024). ICDXML: Enhancing ICD coding with probabilistic label trees and dynamic semantic representations. Scientific Reports, 14, 18319. https://doi.org/10.1038/s41598-024-69214-9
[13] Denecke, K., May, R., & Rivera-Romero, O. R.-R. (2024, February 17). Transformer Models in Healthcare: A Survey and Thematic Analysis of Potentials, Shortcomings and Risks [Review of Transformer Models in Healthcare: A Survey and Thematic Analysis of Potentials, Shortcomings and Risks]. Journal of Medical Systems. https://pmc.ncbi.nlm.nih.gov/articles/PMC10874304/pdf/10916_2024_Article_2043.pdf

Acknowledgements

This class file was developed by Sunrise Setting Ltd, Brixham, Devon, UK.
Website: http://www.sunrise-setting.co.uk