Title: Predicting Emergency Room (ER) Readmissions within 72 Hours of Discharge
Authors: Hershel Mehta, Maeva Fincker, Sara Altman

Introduction:
We worked with an Electronic Health Records (EHR) dataset to predict if a patient will return to the Emergency Room (ER) within 72 hours of discharge from the hospital. A return within 72 hours of discharge suggests that the patient should probably not have been discharged. Therefore, because new admissions are costly -- both in terms of finances and other resources (hospital beds, etc.) -- it would be useful for hospitals to be able to predict whether a patient is likely to return to the ER within 72 hours when considering discharging the patient. In our dataset, there are a total of 71,283 patients who returned to the ER within 72 hours of discharge.

Data and features
We use the *STRIDE-7* dataset, which includes patient EHR data from Stanford Hospital Center (SHC) and Lucile Packard Children Hospital (LPCH). We only use data from SHC, since we are mostly interested in predicting 72-hour readmissions amongst an adult population.

Each patient's EHR data consists of a set of chronological visits to the hospital, where the following data is collected: medication codes, procedure codes , and diagnostic codes. We used EHR codes to create a set of features.

We created our labeled dataset consisting of visits with a discharge time that were either followed by another visit to the ER within 72 hours (label = 1), or visits with discharge time that were not followed by another visit to the ER within 72 hours (label = 0). Since there were far more negative labels than positive labels, we included 9 negative examples (i.e., instances in which a patient did not return to the ER within 72 hours of discharge) for each positive example.

Baseline models
We created three baseline models: a logistic regression model, an SVM, and a multilayer perceptron.

Data
To fit these models, we extracted EHR data from the past 6 months prior to each visit. We did not use a patient's full history as we thought it was likely to be noisy in terms of predicting an outcome within a short time frame (i.e., 72 hours). We then created a sparse set of features consisting of the number of counts of each code (i.e., medical diagnosis and procedure codes) in the 12 months prior to the ER visit. See the table below for a representation of our data in all baseline models.

Table 1: Data format for baseline models

patient_id	visit_id	label	Code 1	Code 2	...	Code N
111	1	1	0	3		0
111	2	0	0	0		1
222	1	1	0	0		0

Following the steps in [2], we pruned away features (i.e., codes) that occurred in less than 100 patients, to prevent an exploding number of features, and ended up with ~1000 features. We used a 80-20 training-test split to fit our models.

Preprocessing

For the Logistic Regression and SVM, we reduced the feature space using PCA after center and scaling. All baseline models were trained with the PCA transformed data. We left the number of components to keep in the subsequent models as a hyperparameter.

Logistic Regression

For the logistic regression model, we used L2 regularization and used 3-fold cross-validation on the training set to select the number of PCA components to keep as well as the L2 regularization parameter. We used F1 score as the cross-validation metric in all our models since accuracy is not informative when classes are unbalanced. Furthermore, both precision and recall should be balanced for our application. The training F1 score was 0.47 and the test F1 score was 0.48.

SVM

Our second baseline model in an SVM with a radial basis function kernel. Hyperparameters for the SVM model were similar to the ones for logistic regression; we used the default value of 1/n_feature for the Rbf kernel parameter. The model never predicted that a patient would return within 72 hours, thus training and test F1 scores are both 0, and test accuracy is the proportion of negative examples in the test set (.88). SVM's can perform poorly with unbalanced classes, and our model likely would have performed better if we had used a weighted loss.

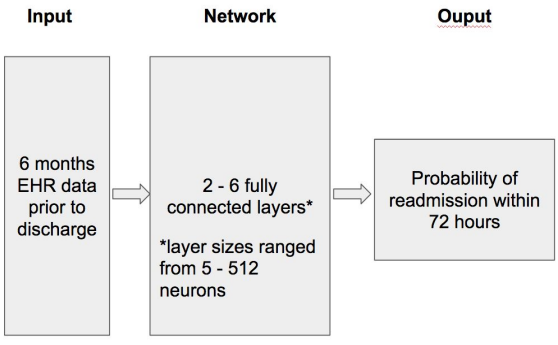


Fig. 1: Dense model architecture

Dense model

Our final baseline model was a fully-connected neural net (which we called our "dense" model). We tuned our model with a ReLU at each hidden layer and a Sigmoid at the output layer, a learning rate of 0.001, an Adam optimizer, a batch size of 512, and an optimized L2 regularization. We also

experimented with different numbers of layers (from 2 - 6 layers) and neurons in each layer (from 5 - 512), and found our best “dense” model had 4 layers with configuration of [50, 25, 10, 5] neurons in layers [1, 2, 3, 4] respectively. Our optimal “dense” model had an F1 score of 0.56 on positive examples and an accuracy of 0.91.

RNN

While dense models ignore the sequential nature of EHR data, Recurrent Neural Networks should be able to learn temporal patterns. We therefore implemented an RNN architecture in Tensorflow to test whether temporal information could help prediction.

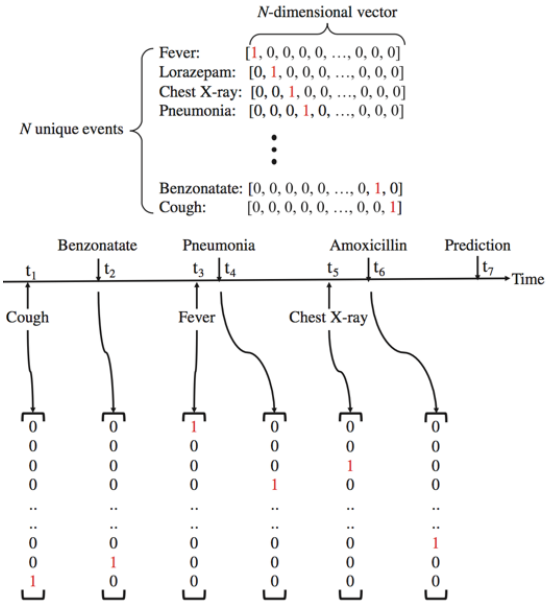


Fig. 2: Data encoding and processing steps for the recurrent neural network architecture

Data extraction and preprocessing

For every visit in our dataset, EHR codes from the previous 6 months before discharge were extracted and sorted chronologically. Examples were split 80:10:10 into train:dev:test sets. Codes that were too common (appearing more than 150,000 times in the dataset) were filtered out with the idea that they would act as “Stop words.” We kept the next 10,000 most common codes. Although RNN can take as input a code sequence of an arbitrary length, we were limited by our access to computing power and therefore had to cap the length of the code sequence. The maximum code sequence length *max_length* was left as a hyperparameter. At run time, each code sequence was either clipped or zero-padded to the fixed *max_length* length. Each code was then encoded as a 10,000 dimensional one-hot vector before being fed to the network. Fig. 2 provides an example of the data encoding and processing steps.

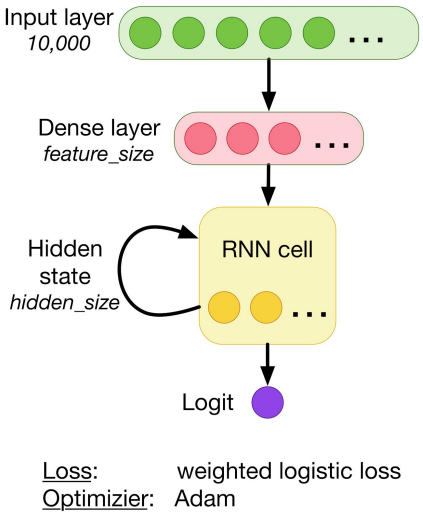


Fig. 3: RNN model architecture

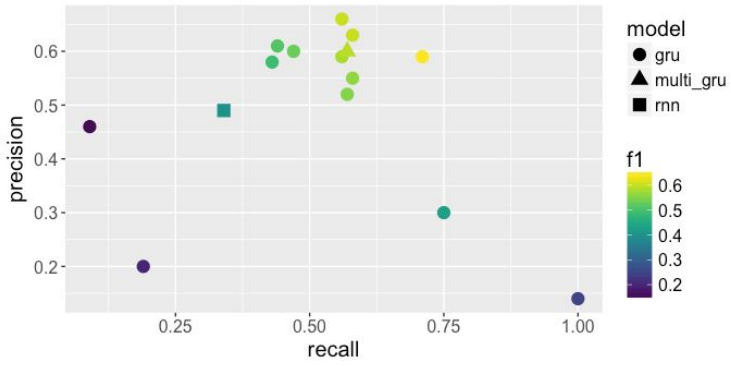
Model architecture:

Our model is composed of a fully connected layer followed by an RNN and then an output layer. The role of the first fully connected layer is to embed the sparse one-hot vector into a smaller feature space in order to facilitate learning of the RNN. The hidden state of the RNN on the last time-step corresponding to a code (and not a 0 from sequence padding) was used as input for the last layer, which implements a sigmoid activation function (Fig. 3). To fight class imbalance, we used a weighted logistic loss with a tunable *positive_weight* parameter. The loss was minimized with an Adam optimizer and the model trained for a maximum of 50 epochs. After every epoch, the model was evaluated against

the dev set and the model with best F1 score against the dev set was saved.

Hyperparameter tuning

Out of all the possible hyperparameters to tune, the most important ones for our models were the learning rate and the weight for positive examples in the loss. Additionally, we tuned the maximum code sequence length, the embedding size of the fully-connected layer, the type of RNN cell and size of its hidden state. Because of time and computing limitation, we could not try code sequences longer than 500 codes and had to downsample our dataset to 5000 examples during hyperparameter tuning. Table 2 lists the different hyperparameter values that were tested and figure 4 displays 3 metrics from the trained models. Once the best hyperparameter values were established, the model was retrained on the full dataset (~150,000 examples).



trained during hyperparameter tuning

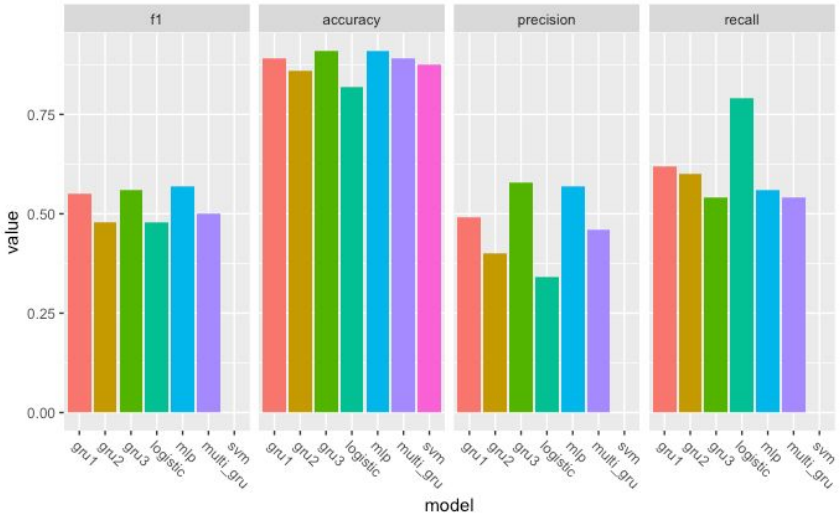
Table 2: Hyperparameters tested - value for the best model are in bold.

learning rate	hidden_size	feature_size	max_length	model	pos_weight
0.1	10	100	10	**gru**	1.5
0.001	**50**	500	**50**	rnn	5
	100	**1000**	100	multi_gru	**7**
			500		10

Fig. 5: Model comparisons

Model comparisons

The 3 best RNNs, the best dense model as well as the SVM and logistic regression were evaluated against the test set and their accuracy, precision, recall and F1 score are reported in figure 5. The dense model (“mlp” in the figure) and the GRU1 and GRU 3 perform



similarly and tie for best models depending on the desired trade off between precision and recall.

Conclusions

We tried to predict admission to the ER within 72h of hospital discharge using EHR data. We compared traditional ML classifiers (SVM, logistic regression) with two different deep-learning architecture (dense models and RNN). The neural networks were more expressive and yielded better F1 score than the traditional models. Furthermore, RNNs are doing as good as the dense model with less data (50 codes vs. 6 months worth of EHR codes). However, precision and recall are still too low to make the models useful in the hospital setting.

Future work

We consider a few areas for future work. First, it would likely improve our results to reduce the dimensionality of our features by learning efficient representations of medical concepts, as demonstrated in [5]. Second, we could incorporate lab tests and their results. We currently only use medication, procedure, and diagnosis codes as features, but our data also includes lab data. To counteract the variance problem in our RNNs (overfitting of the training data after 10 epochs), we could use more training data, increase L2 regularization, or try dropout. Finally, the fact the RNNs do as well as dense models with much less data suggests that the RNNs would outperform the dense model if given access to a similar amount of data. Therefore, we would try to train RNN models on longer code sequences.

Citations

[1] Clinical Classifications Software (CCS) for ICD-9-CM. Agency for Healthcare Research and Quality. https://www.hcup-us.ahrq.gov/tools software/ccs/ccs.jsp. Accessed February 2018.

[2] Improving Palliative Care with Deep Learning. (2017). Avati et al. arXiv: 1711.06402.

[3] Using recurrent neural network models for early detection of heart failure onset. (2017). Choi E, Schuetz A, Stewart WF, Sun J. *J Am Med Inform Assoc*. 24(2):361–70

[4] Choi E, Bahadori MT, Searles E, Coffey C, Thompson M, et al. 2016. Multi-layer Representation Learning for Medical Concepts. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1495-1504

Code Repo

https://github.com/mfincker/ehr_deeplearning

Contribution

Hershel*:*

Data processing: extracting codes from stride7, creating sparse feature matrix, Dense model architecture

Maeva:

Data preprocessing: defining labels / aggregation for dense model
RNN: data formatting / implementation / training / evaluation

Sara:

SVM / Logistic regression baseline
Feature engineering (Med2Vec)