IMPLEMENTATION GUIDE

Health & Life Sciences
Data Driven

Model For Predicting Rapid Response Events

This implementation guide provides key learnings and configuration insights to integrate technologies with optimal business value.

If you are responsible for…

- **Business strategy:** You will better understand how implementation options can impact business capabilities.
- **Technology decisions:** You will learn about available technology alternatives, proven methods to mitigate risk, and techniques to maximize value before starting a project.

Introduction

This document details the steps and processes required to implement the described Model for Predicting Rapid Response Events.

This predictive model was developed as an example use case for the practical application of predictive analytics in a healthcare delivery organization. This implementation guide provides an overview of the process to re-implement this model. Model performance may vary based on unique attributes of local data. Users should consider this implementation guide as a starting point only and plan for work to validate model performance and applicability based on local factors. This model is intended for hospital operations use only. It is not intended for clinical care.

Background

In recent years, increasing numbers of hospitals have implemented Rapid Response Teams (RRTs), also known as Medical Emergency Teams (METs). These teams of medical personnel focus on quickly assessing and treating hospital patients who are experiencing an often-sudden decline in health. These situations, referred to as RRT events, indicate that a patient is experiencing unusual difficulties and needs immediate attention.

There are several existing methods which use a single score to assess patient status. Two measures used today are the Rothman Index and the Modified Early Warning System (MEWS).

This project sought to identify a new method for developing an inpatient risk scoring system by applying machine learning to electronic medical record data. We used RRT events as the modeling target. Based on available data, we predicted the probability that a patient will have an RRT event within 1 hour. Among the distinguishing features of this work is that it addresses both interaction effects and patient characteristics. The intent is that by deploying risk scores, hospital personnel will have advanced notice of ailing patients, which will allow for quicker human response and deployment of medical resources.

The project presented below documents the process for developing a risk score model using inpatient data from a leading electronic medical records (EMR) system at a single hospital to identify patients at risk of RRT events within a defined window of time. Correspondingly, the scoring model should be evaluated for suitability before deployment with another hospital or patient population.

Table of Contents

Solution Overview

Basic Components

The basic components of the solutions discussed in this paper include:

- EMR system – This includes electronic medical records (EMR) such as patient vital signs, demographics, medications, and most importantly, well-documented records of RRT events.
- Cloudera* Cluster environment – This is where the EMR data is uploaded.

Architecture and Workflow

EMR data were ingested via Sqoop* into a Cloudera Cluster environment and stored as Hive* tables (Figure 1). Impala* (via the Hue* interface) was used to initially explore the data, and Jupyter* notebooks were used to examine and model data in a Python* environment. We used a machine-learning model to calculate risk scores on new patient data and stored those patient scores as a new Hive table for reporting purposes.

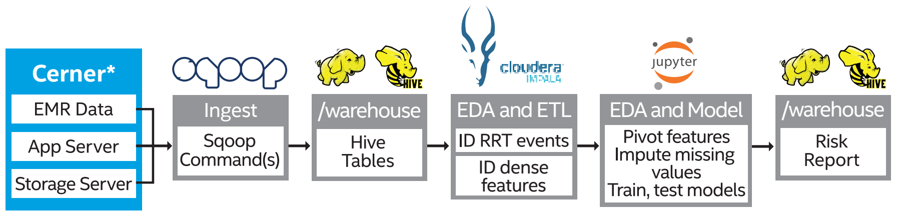

Figure 1. Basic components of the solution architecture.

Solution and System Requirements

Requirements to complete this project include:

EMR System

- For this project, a Cerner* EMR system was used. Other EMR systems could be used, but the user will need to identify the fields and codes in their system that correspond to the fields and codes that are used in this guide.
- Please have data model experts verify that the fields are what they should be. Even if the same EMR is used, the details of the implementation may vary from the data tables and fields shown here.
- Patient vital signs, demographics, medications, and well-documented records of RRT events are expected. In our data model, we relied most heavily on the following tables:
  - `clinical_event` – This is main table that contains entries for each action taken during a patient's hospital stay.
  - `code_value` – This table decodes the values captured in clinical_event.event_cd, and other coded fields.
  - `encounter` – This provides details about each patient's individual visits to the hospital.
  - `patient` – This contains information on patient characteristics (age, sex, race, and so on).
  - `mltm_drug_categories, orders, mltm_category_drug_xref` – These tables contain information on drugs ordered for patients.
- Patient data needs to have sufficient density. Healthier patients do not generally have vitals recorded as frequently as less healthy patients, yet these are important data to have for both patients with and without RRT events. The results described in this guide were obtained from data collected every four hours.
  - We chose a 12-hour window (13 hours prior to the 1 hour before an RRT event) as the time interval from which we would select data to build the features for modeling. The target was a binary label, 0 or 1, representing whether the

patient had an RRT event between hour 12 and hour 13. The decision to use a 12-hour window was based on exploratory data analysis that allowed us to assess the sparsity and prevalence of the vitals data of interest. We suspect using an 8- or 6-hour time window may lead to an improved model.

- How much data is needed?
  - Preliminary results were achieved with about 2000 examples each of the positive and negative classes. These records came from one hospital during a 1.5-year timeframe.
  - We found that extending the historical window to include more records provided more training examples. These additional training examples improved the performance of our predictive model.
  - It is important to note that procedures for recording data about RRT events may change over time. Work with your data model experts to identify any such changes in record keeping that may not be obvious from the data, and determine the proper timeframe for sample data.

Cloudera* Cluster Environment

Cloudera Hadoop* Cluster (CDH version 5.x), with the following components:

- HDFS*
- Hue
- Hive
- Impala
- Anaconda Python v2.7 installation with Jupyter notebooks (package dependencies are specified in **`patientrisk.yml`** in Appendix A)
- 4th generation Intel® Xeon® processor E5 family

Data Understanding and Analysis

Data Examined

Significant data exploration may be needed to uncover the ideal set of data to work with. We provide amplifying discussion, example queries, and helper functions in the Appendices. A more comprehensive resource is available as a GitHub* repository at github.com/bartleyintel/model-for-predicting-rapid-response-team-events. This repository contains documentation, Jupyter notebooks, and copies of SQL queries that can be executed in Impala to recreate the analysis and work that generated this use case.

Initially, we explored patients with RRT events and their characteristics. Then we explored the data we suspected of being predictive of RRT events. Through the exploratory data analysis process and many meetings with subject matter experts (SMEs), we arrived at a list of 27 features used for modeling purposes. Of these, 14 were calculated from vital signs (7 values to represent the mean of the vitals data and 7 to represent the most recent values of vitals) over the time frame of interest.

Example Impala* Query to Find Encounters with RRT Events

We had to select the right kinds of patient records to use for this analysis. Here is an annotated query we used to select people with Rapid Response Team (RRT) events:

```
SELECT enc.encntr_id
FROM encounter enc
INNER JOIN clinical_event ce ON enc.encntr_id = ce.encntr_id
WHERE enc.loc_facility_cd='633867'          -- Patients from the facility of interest
  AND enc.encntr_complete_dt_tm < 4e12      -- Patients with complete encounters
  AND    ce.event_cd='54411998'             -- Event code indicating an RRT Event
  AND ce.result_status_cd NOT IN ('31', '36') -- Codes for invalid result status codes
  AND    ce.valid_until_dt_tm > 4e12        -- Times greater than 4e12 are invalid events
  AND ce.event_class_cd not in ('654645')   -- Codes that represent invalid date/times
  AND    enc.admit_type_cd !='0'            -- Only use admitted patients
  AND enc.encntr_type_class_cd='391'        -- Inpatients only
ORDER BY enc.encntr_id, event_end_dt_tm
```

The subset of data we were most interested in could easily fit in memory in the master node for the cluster. We used the Python package Pandas* to analyze, and scikit-learn* to model the data in Jupyter notebooks. If these is more data than will fit in memory, it may be necessary to use a tool like Spark* and rely on SparkML* for modeling.

Notes on Data for Understanding and Analysis

We found that we needed to add many clauses to choose the right type of patients for this analysis. We have included an example query to find encounters with RRT events. The query contains several different WHERE clauses—these conditions were discovered after discussions with SMEs and investigating the data.

Preparing your Data for Modeling

Besides determining which records to query and how to query only valid records, there are several options available for preparing your data for modeling.

Time frame

- We chose a 12-hour time frame during which we calculated the mean and most recent vital signs, and determined whether one of several medication classes was prescribed. We also found or calculated patient conditions like smoking and obesity. This summarized data was used to calculate the probability the patient would have an RRT event in 1 hour.
- We assessed the impact of data capture techniques (manual vs. telemetered data) on data density to determine sample period.
- The 12-hour time frame was chosen due to data density. A smaller time window may have fewer data points available but may provide better predictions.

Feature selection

- Interviews with SMEs may identify features that cannot be included due to lack of data or data sparsity. It may be necessary to make judgements and drop potential features which lack enough data for modeling. One best practice is to document these features and assess a potential action plan to increase data collection to improve model performance in the future.
- Adding more features means more training data are needed. Beware if training data is limited, as increasing features without a corresponding increase in the number of training instances will result in increased modeling error due to variance.
- To ensure utility of any created models, a good rule of thumb is to ensure that the data used to create a model is as close as possible to the data that will be available in the production environment. This means that features should only be used that are likely to be present in most patients that the model is intended to make predictions for.

Choosing how to parameterize medications

- We looked within a 12-hour time frame to determine what medications the patient was prescribed.
- We classified medications by broad categories such as opioids, chemotherapy, and anti-psychotics to keep the data dimensionality low. Categorizing features inherently loses information about individual drugs, but we thought it was a reasonable trade-off given the small number of training instances.
- Some other potentially useful ways to represent medication concepts are to:
  - Use the dosage of different medications in a time frame to represent how much of each medication is present in a certain time interval.
  - Count how many medications the patient was taking at the time of admission. Clinicians have expressed that this information informs their intuition about how likely a patient is to have complications during their encounter.
  - Count the total number of medications taken since the beginning of the encounter to assess the degree to which a patient has been medicated since being admitted into the hospital.
  - Count the number of distinct medications taken within a time interval to try to capture effects due to drug interactions.

Cohort selection

- Decide which patients to include in the cohort for the analysis. One option is to only select patients admitted to the emergency room rather than all inpatients. You may not want to use a mixture of non-emergency and emergency patients as you might expect a different distribution of RRT events between these patient groups. However, we opted to include inpatients with Urgent, Emergency, and Elective classifications, to include as much of the limited data as we could. In the data we worked with, there were roughly even numbers of Urgent and Emergency patient types.

Data Examined but not Included in Modeling

Potentially, there are many other features that could be predictive. Be aware that there is a trade-off wherein each feature added to the modeling process increases the amount of training data required. Some other possible features include:

- Changing locations of patients between departments with different care levels (e.g., Emergency to ICU)
- Lab test results
- Lactic acid or lactate levels
- Potassium levels
- LVAD (Left Ventricular Assist Device, which is implanted to support a failing heart)
- Medication stacking (number of unique medications taken)
- Nursing assessments

Modeling Process

After a thorough examination of the data available, and after making a preliminary choice regarding the features to use for modeling, we can start the modeling process. See Figure 2 for a graphical overview of the predictive modeling process.

Selecting and Preparing Data

Data from patients with and without RRT events are required for this work. Patients with RRT events are the limiting data here. There will be far more patients without RRT events, though the data quality regarding these patients may vary significantly – healthy people tend to have fewer vital signs recorded in their EMR data.

The notebook `create_modeling_table.ipynb` shows how we used Impala and Python to query data and shape it into a tabular format appropriate for modeling. Categorical data may need to be recast from string or varchar data types to numeric data that the modeling algorithm can use. We decided to not normalize and rescale the data. This means that modeling results will be more interpretable. However, it is possible to normalize and rescale your data to improve model performance.

Missing Data

An important modeling and analysis question is how to handle missing or incomplete data. There are several ways to handle missing data that can take on continuous values:

- Impute the missing value with the mean or median of that column.
- Use another model, such as linear regression, to predict the missing values based on the features that are present.

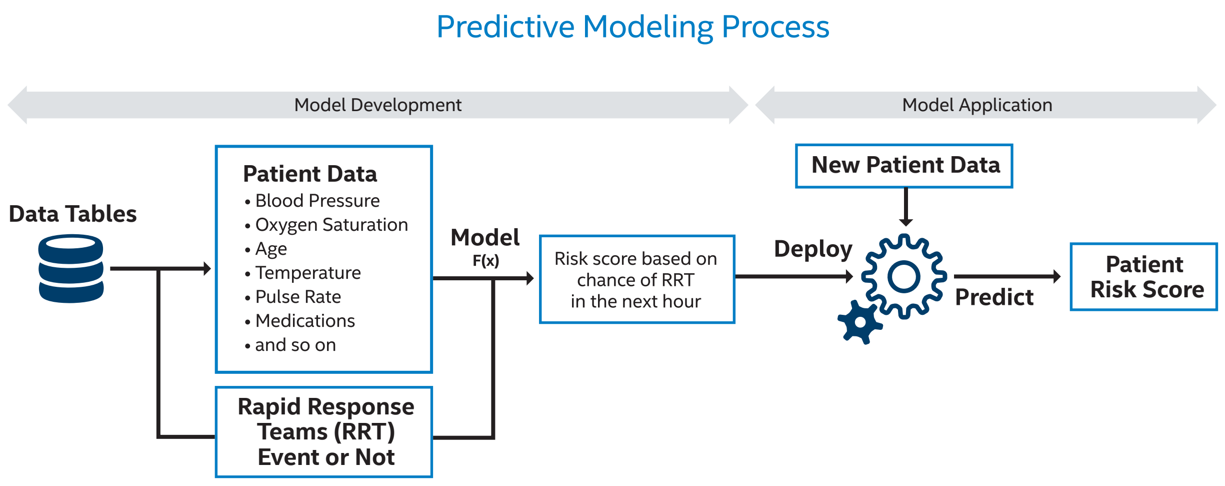

Figure 2. Schematic describing the predictive modeling process

- The most extreme case: drop the rows or columns that contain excessive missing values. **NOTE:** When considering whether to drop rows with missing data, it is important to try and determine if certain rows are missing at random or not at random. This is important because data that is missing (randomly or not) is missing for a reason, and removing those rows may introduce bias into your cohort selection. One way to test for this is to conduct a statistical test for the difference in distribution of RRT events between the missing and non-missing valued rows.

In developing this model, we first dropped data with excessive missing values, then imputed any individual missing values using the mean with the average of all the patients in the training data (both with and without RRT events).

Outliers

In this project, we did not include screening for bad data points – some sort of outlier detection may be useful in checking that the available data is of good quality.

Model Selection and Application

After fitting several models and setting hyper parameters, we used a gradient-boosted decision trees classifier acting on cleaned and unscaled data, as shown in **`modeling_base.ipynb`**. Use the notebook, **`modeling_diff_algorithms.ipynb`**, to see a demonstration of how we tried and compared models created with different algorithms.

If the data remains unscaled, then partial dependence plots can give the user information on how changes in vital signs affect the probability of having an RRT event, which is useful. We found that scaling the data did not have a significant impact on our final results, thus we used unscaled data for easier interpretability.

Model Validation

We perform 5-fold cross-validation on a training subset of the original data, after performing a grid search to find optimal modeling parameters. The training subset was chosen to be 70 percent of available data; the 30 percent of hold-out data was used to evaluate the final model.

We used AUC-ROC (Area Under the Receiver Operating Characteristic Curve) as the binary classification metric to optimize. A confusion matrix is helpful to visualize model performance with regards to false positives and false negatives. It is possible to tune the model parameters in response to the number of false positives and false negatives seen.

It is helpful to look at the resulting histogram of the prediction probabilities to see if the distribution is strongly skewed. A good binary classifier will result in a U-shaped distribution because most predictions are concentrated close to 0 or 1, which is exactly what we want.

In our proof of concept, we translated the model output (the probability of developing an RRT event within 1 hour) into a risk score by multiplying by 10 and rounding. This produced a risk report where most patients were concentrated near either 0 (least risky) or 10 (most risky). In a deployed version, the risk score may be more useful if the values are normalized to be more uniform.

Notes on Modeling

- We determined the number of patients who had RRT events, then selected approximately the same number of patients without RRT events who had similar patient characteristics. This sample of non-RRT event patients likely does not cover the full range of variability those patients experience. A better approach may be to increase the number of non-RRT event patients, and up-sample the patients with RRT events to achieve class balance.
- A rule of thumb is to increase the dataset by a minimum of 10 additional examples for each feature in the modeling dataset. Although, this will likely result in a classifier that is mediocre. Achieving a level of classification performance that is useful in an operational setting will likely require many more examples than 10 per feature, possibly several hundred per feature. This situation is a consequence of the Bias-Variance Trade-off.

Deployment

Before considering deployment, we recommend a trial or pilot where questions about the consumption and validity of the scoring approach are examined.

There are several factors to consider:

- Where will the scores reside, how will they be consumed by staff?
  - What clinical and business workflows will need to change in order to activate the appropriate actions based on model results?
- How often should scores be recalculated when new data is available?
  - Perhaps scores only need to be calculated during shift changes, for example, when fresh staff take over and need to be brought up to date quickly on patient status.
- How will model performance and staff interaction be captured and used to improve model performance over time?
- In medicine, a randomized controlled trial (RCT) is the gold standard to show that a given intervention has an impact over another intervention or the status quo. These RCTs can be costly in both time and money, but are essential in medicine. To develop an RCT for this project:
  - Determine the objective evaluation criterion – that is, what is the agreed upon measure of success? Are we trying to reduce the average number of RRT events per day? Or are we trying to reduce the workload of the nursing staff?
  - Statistical power for the RCT needs to be calculated so there is a good chance of detecting an effect, if it exists. This translates to the length of time the experiment would need to run to observe a certain effect size. Given that RRT events are rare, this may take weeks or months.
  - Random assignment – patients should be assigned randomly to the control (no risk scoring model) or the intervention (risk score available). The patients should otherwise be treated similarly.
  - Data must be collected in a uniform manner for all patients in the RCT, regardless of which group the patient is assigned to.
- A classification model can be made more – or less – conservative by adjusting the default decision threshold. This is done by extracting the prediction probabilities and applying a decision threshold different than the default threshold of 0.5 – probabilities less than 0.5 are predicted to be non-RRT events and probabilities greater than 0.5 are predicted to be RRT events. For example, if a hospital's operational needs require fewer false positives, then it is simply a matter of filtering out predicted probabilities less than 0.8 (or some other number between 0 and 1). While this will surely result in fewer "false alarms," the trade-off is failing to identify more patient RRT events (false negatives) and possibly having less advanced notice of the RRT events that are detected. Managing the trade-off between false negatives and false positives is one of the key decisions that needs to be addressed in operationalizing any predictive model.

Summary

In this document, we outlined the steps taken to develop a patient risk score for hospital inpatients. These risk scores are built from data on RRT events experienced by hospital inpatients. With appropriate data, this document can be used as an outline to develop a patient risk scoring model based on your own data.

Appendix A: Python* Packages and Environment

More detailed information is available on the project GitHub page. There are folders containing Impala* queries used for querying the data, and Jupyter* notebooks on Exploratory Data Analysis and Modeling.

We used the Anaconda* scientific Python distribution using Python* v2.7. Below are the contents of a .yml file containing packages important to the project. Copy the lines below, beginning with "name:" and paste them into a file such as **`my-environ.yml`**.
The environment can be loaded by using the command: **`conda env create -f my-environ.yml`**.

```
name: modelforpredictingRRTevents
channels: !!python/tuple
- !!python/unicode 'defaults'
dependencies:
- bitarray=0.8.1=py27_0
- cycler=0.10.0=py27_0
- freetype=2.5.5=1
- icu=54.1=0
- impyla=0.13.8=py27_0
- libpng=1.6.22=0
- matplotlib=1.5.3=np111py27_1
- mkl=11.3.3=0
- numpy=1.11.2=py27_0
- openssl=1.0.2j=0
- pandas=0.19.1=np111py27_0
- pip=9.0.1=py27_0
- pyparsing=2.1.4=py27_0
- pyqt=5.6.0=py27_0
- python=2.7.12=1
- python-dateutil=2.5.3=py27_0
- pytz=2016.7=py27_0
- qt=5.6.0=0
- readline=6.2=2
- scikit-learn=0.18=np111py27_0
- scipy=0.18.1=np111py27_0
- seaborn=0.7.1=py27_0
- setuptools=27.2.0=py27_0
- sip=4.18=py27_0
- six=1.10.0=py27_0
- sqlite=3.13.0=0
- thrift=0.9.3=py27_0
- tk=8.5.18=0
- wheel=0.29.0=py27_0
- zlib=1.2.8=3
- pip:
  - docopt==0.6.2
  - funcsigs==1.0.2
  - hdfs==2.0.13
  - hmmlearn==0.2.0
  - ibis-framework==0.8.1
  - mock==2.0.0
  - pbr==1.10.0
  - py==1.4.31
  - pytest==3.0.3
  - requests==2.11.1
  - sqlalchemy==1.1.3
  - toolz==0.8.0
```

Appendix B: Features Created and Data Source

More detailed information is available on the project GitHub page. There are folders containing Impala queries used for querying the data, and Jupyter notebooks on Exploratory Data Analysis and Modeling.

Feature Name	Feature Type	Cerner* Table	Cerner Field
Mean Arterial Pressure (MAP)	Vital Sign	clinical_event	event_cd = 703306
Systolic Blood Pressure (SBP)	Vital Sign	clinical_event	event_cd = 703501
Peripheral Pulse Rate (pulse)	Vital Sign	clinical_event	event_cd = 703511
Diastolic Blood Pressure (DPB)	Vital Sign	clinical_event	event_cd = 703516
Respiratory Rate (RR)	Vital Sign	clinical_event	event_cd = 703540
Temperature Oral (temp)	Vital Sign	clinical_event	event_cd = 703558
Height/Length (obese)	Vital Sign	clinical_event	event_cd = 2700653
SpO2 (SPO2)	Vital Sign	clinical_event	event_cd = 3623994
Measured Weight (obese)	Vital Sign	clinical_event	event_cd = 4674677
Smoking Code (smoker)	Patient Info	clinical_event	event_cd = 75144985
On IV Indicator (on_iv)	Patient Info	clinical_event	event_cd = 679984
Buprenorphine-Naloxone (bu-nal)	Medication	clinical_event	event_cd = 2797130
Naloxone (bu-nal)	Medication	clinical_event	event_cd = 2798305
Buprenorphine (bu-nal)	Medication	clinical_event	event_cd = 2797129
Narcotic Analgesic (narcotics)	Medication	mltm_drug_categories	multum_category_id = 60
Narcotic Analgesic Combination (narcotics)	Medication	mltm_drug_categories	multum_category_id = 191
Antipsychotics (antipsychotics)	Medication	mltm_drug_categories	multum_category_id = 77, 210, 251, 341
Chemo Drugs (chemo)	Medication	mltm_drug_categories	multum_category_id = 20, 21, 22, 23, 24, 25, 26
Anticoagulants (anticoagulants)	Medication	mltm_drug_categories	multum_category_id = 261, 262, 283, 285
Age	Patient Info	person	age
Sex (is_male)	Patient Info	person	sex

Appendix C: Exploring Reasons for Rapid Response Team Events

More detailed information is available on the project GitHub page. There are folders containing Impala queries used for querying the data, and Jupyter notebooks on Exploratory Data Analysis and Modeling.

The following SQL queries the counts of the most frequent reasons for RRT events. An example query has been provided as a template where you need to edit the table name, column names, and where clauses to make them appropriate for your database schema.

```
SELECT ce.event_tag, COUNT(*) as num_occurrences
FROM encounter enc
INNER JOIN clinical_event ce
ON enc.encntr_id = ce.encntr_id
WHERE enc.loc_facility_cd='633867'            -- Specify the facility location of interest
AND enc.encntr_complete_dt_tm < 4e12          -- Filter out incomplete/invalid events
AND ce.event_cd='54408578'                    -- Use the code for RRT events reasons
AND ce.result_status_cd NOT IN ('31', '36')   -- Filter out code for corrupted/bad events
AND ce.valid_until_dt_tm > 4e12               -- Filter out incomplete/invalid events
AND ce.event_class_cd not in ('654645')       -- Code for time based errors
AND enc.admit_type_cd !='0'                   -- Ensure we are selecting admitted patients only
AND enc.encntr_type_class_cd='391'            -- Ensure we are selecting inpatients only
GROUP BY ce.event_tag
ORDER BY num_occurrences DESC
LIMIT 30
```

Appendix D: Exploring Encounter Durations

More detailed information is available on the project GitHub page. There are folders containing Impala queries used for querying the data, and Jupyter notebooks on Exploratory Data Analysis and Modeling.

The following SQL query queries the patient encounter durations. An example query has been provided as a template where you need to edit the table name, column names, and where clauses such to make them appropriate for your database schema.

```
SELECT DISTINCT
    ce.encntr_id,
    COALESCE(tci.checkin_dt_tm, enc.arrive_dt_tm) AS checkin_dt_tm,
    enc.depart_dt_tm as depart_dt_tm,
   (enc.depart_dt_tm - COALESCE(tci.checkin_dt_tm, enc.arrive_dt_tm))/3600000 AS diff_hours,
    enc.reason_for_visit, enc.admit_type_cd, cv_admit_type.description as admit_type_desc,
    enc.encntr_type_cd, cv_enc_type.description as enc_type_desc,
    enc.encntr_type_class_cd, cv_enc_type_class.description as enc_type_class_desc,
    enc.admit_src_cd, cv_admit_src.description as admit_src_desc,
    enc.loc_facility_cd, cv_loc_fac.description as loc_desc,
FROM clinical_event ce
INNER JOIN encounter enc ON enc.encntr_id = ce.encntr_id
LEFT OUTER JOIN code_value cv_admit_type ON enc.admit_type_cd = cv_admit_type.code_value
LEFT OUTER JOIN code_value cv_enc_type ON enc.encntr_type_cd = cv_enc_type.code_value
LEFT OUTER JOIN code_value cv_enc_type_class ON enc.encntr_type_class_cd = cv_enc_type_class.code_value
LEFT OUTER JOIN code_value cv_admit_src ON enc.admit_src_cd = cv_admit_src.code_value
LEFT OUTER JOIN code_value cv_loc_fac ON enc.loc_facility_cd = cv_loc_fac.code_value
LEFT OUTER JOIN  (  SELECT ti.encntr_id AS encntr_id,
                    MIN(tc.checkin_dt_tm) AS checkin_dt_tm
                    FROM tracking_item ti
                    JOIN tracking_checkin tc ON ti.tracking_id = tc.tracking_id
                    GROUP BY ti.encntr_id
                 )  tci
ON tci.encntr_id = enc.encntr_id
WHERE enc.loc_facility_cd = ‘633867’
    AND enc.encntr_complete_dt_tm < 4e12
    AND enc.admit_type_cd!=’0’
    AND enc.encntr_type_class_cd=’391’
    -- Currently select patient with RRT events.
    AND enc.encntr_id IN (  SELECT DISTINCT ce.encntr_id
                          FROM clinical_event ce
                          WHERE ce.event_cd = ‘54411998’     -- This is the RRT event code
                            AND ce.result_status_cd NOT IN (‘31’, ‘36’)
                            AND ce.valid_until_dt_tm > 4e12
                            AND ce.event_class_cd not in (‘654645’))
```

Appendix E: Exploring Patient Vitals

More detailed information is available on the project GitHub page. There are folders containing Impala queries used for querying the data, and Jupyter notebooks on Exploratory Data Analysis and Modeling.

```
SELECT
    ce.clinical_event_id,
    ce.event_id,
    ce.encntr_id,
    ce.person_id,
    ce.event_cd,
    cv_event_cd.description            AS event_description,
    ce.performed_dt_tm                                     AS unix_performed_dt_tm,
    from_unixtime(CAST(ce.performed_dt_tm / 1000 as bigint)) AS performed_dt_tm,
    ce.event_tag,
    ce.result_val,
    cv_result_units_cd.display       AS result_units_display,
    ce.result_time_units_cd,
    ce.catalog_cd
FROM clinical_event ce
LEFT OUTER JOIN code_value cv_event_cd        ON ce.event_cd = cv_event_cd.code_value
LEFT OUTER JOIN code_value cv_result_units_cd ON ce.result_units_cd = cv_result_units_cd.code_value
WHERE ce.encntr_id = '105479870' -- A randomly chosen patient encounter
AND ce.event_cd IN ('703306',    -- Mean Arterial Pressure
                    '703501',    -- Systolic Blood Pressure
                    '703511',    -- Peripheral Pulse Rate
                    '703516',    -- Diastolic Blood Pressure
                    '703540',    -- Respiratory Rate
                    '703558',    -- Temperature Oral
                    '703565',    -- Glasgow Coma Score
                    '703569',    -- Oxygen Flow Rate
                    '2700541',   -- Heart Rate Monitored
                    '2700653',   -- Height/Length
                    '3623994',   -- SpO2
                    '4674677',   -- Measured Wight
                    '4686698',   -- RASS Score
                    '4690633',   -- CO2
                    '54411998',  -- RRT Event Form
                    '54408578'   -- RRT Primary Reason for Call)
ORDER BY ce.encntr_id, ce.performed_dt_tm
```

Appendix F: Exploring Differences in Vital Signs Between RRT and non-RRT Patients

A Jupyter notebook, `vitals_avg_over_visit[EDA].ipynb`, has been provided that demonstrates an exploratory analysis to determine if the vitals data is obviously different between patients who have RRT events and patients who do not.

The following helper function is useful for creating scatter plots of different vital signs averaged over individual patients and color-coded by if they were RRT or non-RRT patients (see Figure F1).

```
def scatter_avgs(event_cd1, event_cd2):
    '''
    Input: str, event code representing the vital sign of interest.
    Description: This function returns a scatterplot, of the average of the vital signs per encounter.
    '''

    query_RRT = """
    SELECT
        ce.encntr_id,ce.event_cd,
        cv_event_cd.description AS event_description,
        AVG(CAST(ce.result_val as int)) as avg
    FROM clinical_event ce
    JOIN encounter enc ON ce.encntr_id = enc.encntr_id
    LEFT OUTER JOIN code_value cv_event_cd
        ON ce.event_cd = cv_event_cd.code_value
    WHERE ce.encntr_id IN
                    ( SELECT DISTINCT encntr_id
                      FROM clinical_event
                      WHERE event_cd = '54411998'
                      AND result_status_cd NOT IN ('31', '36')
                      AND valid_until_dt_tm > unix_timestamp()
                      AND event_class_cd not in ('654645')
                    )
    AND ce.event_cd IN ('{0}', '{1}')
    AND ce.performed_dt_tm > 1470009600000
    AND ce.performed_dt_tm < 1470355200000
    AND enc.loc_facility_cd = '633867'
    GROUP BY ce.encntr_id, ce.event_cd, cv_event_cd.description
    ORDER BY ce.encntr_id;
    """.format(str(event_cd1), str(event_cd2))

    # Execute the Impala query to get RRT data and merge the dataframes
    cur.execute(query_RRT)
    df_RRT = as_pandas(cur)
    df_1RRT = df_RRT[df_RRT.event_cd==str(event_cd1)]
    df_2RRT = df_RRT[df_RRT.event_cd==str(event_cd2)]
    df_finRRT = pd.merge(df_1RRT, df_2RRT, on='encntr_id')

    query_NotRRT = """
    SELECT
    ce.encntr_id,
    ce.event_cd,
    cv_event_cd.description AS event_description
    AVG(CAST(ce.result_val as int)) as avg
    FROM clinical_event ce
    JOIN encounter enc ON ce.encntr_id = enc.encntr_id
    LEFT OUTER JOIN code_value cv_event_cd
        ON ce.event_cd = cv_event_cd.code_value
    WHERE ce.encntr_id NOT IN
```

```
                    (  SELECT DISTINCT encntr_id
                       FROM clinical_event
                       WHERE event_cd = '54411998'
                       AND result_status_cd NOT IN ('31', '36')
                       AND valid_until_dt_tm > unix_timestamp()
                       AND event_class_cd not in ('654645')
                    )
AND ce.event_cd IN ('{0}', '{1}')
AND ce.performed_dt_tm > 1470009600000
AND ce.performed_dt_tm < 1470355200000
AND enc.loc_facility_cd = '633867'
GROUP BY ce.encntr_id, ce.event_cd, cv_event_cd.description
ORDER BY ce.encntr_id;
""".format(str(event_cd1), str(event_cd2))

# Execute the Impala query to get non-RRT data and merge the dataframes
cur.execute(query_NotRRT)
df_NotRRT = as_pandas(cur)
df_1NotRRT = df_NotRRT[df_NotRRT.event_cd==str(event_cd1)]
df_2NotRRT = df_NotRRT[df_NotRRT.event_cd==str(event_cd2)]
df_finNotRRT = pd.merge(df_1NotRRT, df_2NotRRT, on='encntr_id')

# Create a scatter plot of the data
plt.figure(figsize=(10, 8))
plt.scatter(df_finNotRRT.avg_x, df_finNotRRT.avg_y, s = 40, alpha=0.5, c='blue')
plt.scatter(df_finRRT.avg_x, df_finRRT.avg_y, s = 30, alpha=0.5, c='red')
plt.xlabel(event_cd1)
plt.ylabel(event_cd2)
plt.legend(['Non-RRT patients', 'RRT patients'])
plt.title('Vital sign averages per encounter')

# Create a scatter plot of blood pressure for patients with and without RRT events during their
encounters.
# x = systolic BP; y = diastolic BP
    scatter_avgs('703501','703516')
```

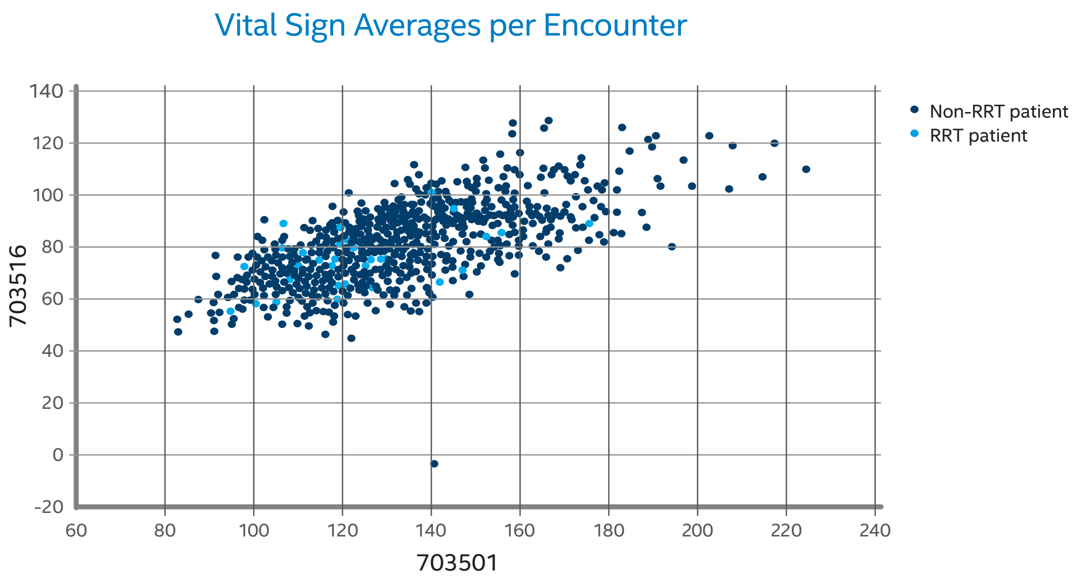

Figure F1. Example of scatter plots of different patient vital signs.

Appendix G: Exploring Patients With Multiple RRT Events

More detailed information is available on the project GitHub page. There are folders containing Impala queries used for querying the data, and Jupyter notebooks on Exploratory Data Analysis and Modeling.

A Jupyter notebook, `multi_rrts[EDA].ipynb`, has been provided to demonstrate an exploratory analysis of patients with multiple RRT events in a single encounter.

Our interviews with clinicians and other SMEs indicated that those who have a single RRT event during a stay are likely to have more RRT events in the future (see Figure G1).

We wanted to identify if somebody had a previous RRT event so that it could be used as a feature in creating a predictive model. However, given our experiences with the messy data of real-world hospital operations, as well as the nuances of Cerner*, we know that determining exactly when something happened is not always a straightforward process.

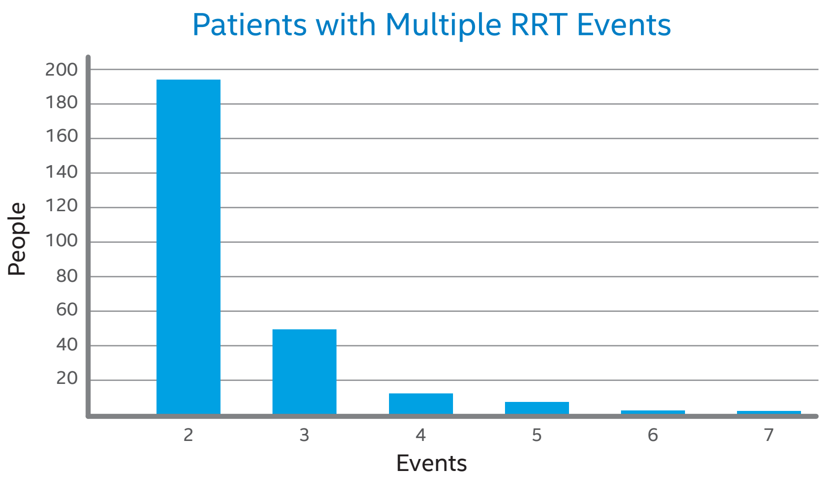

Figure G1. This shows the frequency counts of patients with different numbers of Rapid Response Team (RRT) events during an encounter.

Example of issues that may occur are somebody having an RRT event before an encounter began or after it ended, both of which are impossible. Another situation that may occur is multiple RRT events being logged for a single actual event. For any encounter, if the intervening period between two RRT events was less than an hour, we treated it as a single RRT event.

The following SQL query can be used to query the RRT events for patients who had multiple RRT events during an encounter. The data in this query can be used to determine if there are oddities in the data.

```
SELECT
    ce.encntr_id,
    ce.event_id,
    ce.valid_until_dt_tm,
    from_unixtime(CAST(ce.event_end_dt_tm / 1000 as bigint)) AS event_end,
    ce.event_end_dt_tm,
    from_unixtime(CAST(ce.valid_from_dt_tm/1000 as bigint)) AS valid_from,
    from_unixtime(CAST(enc.arrive_dt_tm/1000 as bigint)) AS enc_arrive,
    enc.arrive_dt_tm,
    COALESCE(tci.checkin_dt_tm, enc.arrive_dt_tm) AS check_in_time,
    from_unixtime(CAST(COALESCE(tci.checkin_dt_tm,
                                enc.arrive_dt_tm)/1000 as bigint)) AS check_in,
    from_unixtime(CAST(enc.depart_dt_tm/1000 as bigint)) AS enc_depart,
    enc.depart_dt_tm,
    FROM clinical_event ce
    INNER JOIN encounter enc
    ON ce.encntr_id = enc.encntr_id
    LEFT OUTER JOIN (
                  SELECT
                     ti.encntr_id AS encntr_id,
                     MIN(tc.checkin_dt_tm) AS checkin_dt_tm
                  FROM tracking_item ti
                  JOIN tracking_checkin tc
                  ON ti.tracking_id = tc.tracking_id
                  GROUP BY ti.encntr_id
                ) tci
```

```
ON tci.encntr_id = enc.encntr_id
WHERE ce.event_cd = '54411998'
AND ce.encntr_id IN (
                    SELECT enc.encntr_id
                    FROM encounter enc
                    INNER JOIN clinical_event ce
                    ON enc.encntr_id = ce.encntr_id
                    WHERE enc.loc_facility_cd='633867'
                   AND enc.encntr_complete_dt_tm < 4e12
                   AND ce.event_cd='54411998'
                   AND ce.result_status_cd NOT IN ('31', '36')
                   AND ce.valid_until_dt_tm > 4e12
                   AND ce.event_class_cd not in ('654645')
                   AND enc.admit_type_cd != '0'
                   AND enc.encntr_type_class_cd = '391'
                   GROUP BY enc.encntr_id
                   HAVING COUNT(*) > 1
                   ) encids
AND ce.valid_until_dt_tm>4e12
```

Appendix H: Exploring Patient Medications

More detailed information is available on the project GitHub page. There are folders containing Impala queries used for querying the data, and Jupyter notebooks on Exploratory Data Analysis and Modeling.

A Jupyter notebook, `medications[EDA].ipynb`, has been provided to demonstrate an exploratory analysis of patient with multiple RRT events in a single encounter. The notebook analysis shows that the proportion of patients taking these drugs is higher in the RRT cohort than the non-RRT cohort.

From interviews with SMEs, we identified a handful of broad categories of medications and drugs that might be useful in predicting patient decline. These categories and their respective Multnum category id codes are shown in the tables below:

Feature Name	Feature Type	Cerner Field
chemo	representing the presence of chemotherapy drugs	multum_category_id= 20, 21, 22, 23, 24, 25
narcotics	representing the presence of narcotics	multum_category_id=60
narc-ans	representing the presence of narcotic analgesic combinations	multum_category_id= 191
anticoagulants	representing the presence of anticoagulants drugs	multum_category_id= 261, 262, 283, 285
antipsychotics	representing the presence of antipsychotic drugs	multum_category_id= 77, 210, 251, 341

Additionally, several other categorical variables were created:

Feature Name	Feature Type	Cerner Field
on_iv	representing if the patient is on an IV drip	clinical event_code= 679984
bu-nal	representing if the patient is taking buprenorphine and/or naloxone	clinical event_code= 2797130, 2797129
dialysis	if the patient is on dialysis	clinical event_code= 186470117

Appendix I: Creating the Modeling Dataset

More detailed information is available on the project GitHub page. There are folders containing Impala queries used for querying the data, and Jupyter notebooks on Exploratory Data Analysis and Modeling.

A pipeline utility has been provided to demonstrate how to query, pivot, and aggregate the patient encounter data into a tabular form so that it can be used in model training and validation.

The pipeline consists of the following steps:

1. Load Impala queries and use the Python impala library to query Hive* for data on the encounters of interest (e.g., electronic chart data, medications, and patient characteristics, and so on). Impala results are cast as Pandas* dataframes.
2. Preprocess the data to include removing duplicate entries, pivoting transactional records of chart events on the Encounter ID or RRT ID, filling missing values, and dropping some columns that are too sparse for use.
3. Concatenate the dataframes on the appropriate common key. The common key is Encounter ID for patients without RRT events and RRT ID for patients with RRT events.
4. Write the results to either a Hive table or a CSV file.

Appendix J: Training, Testing, and Selecting a Model

More detailed information is available on the project GitHub page. There are folders containing Impala queries used for querying the data, and Jupyter notebooks on Exploratory Data Analysis and Modeling.

A Jupyter notebook, modeling_diff_algorithms.ipynb, has been provided to demonstrate how to train, test, and compare the performance of several different modeling approaches.

Handling Missing Data

Before modeling can begin, any missing data must be taken care of. The notebook shows several methods for handling missing data. For some features, the number of rows with missing data are relatively low and they can be imputed using a mean or a median of that feature. In more extreme cases, the overwhelming majority of rows for a column are missing data and that column should be dropped.

Some rows may be missing values for most of the features. In that case, it may be best to drop those rows because they do not have much useful information about them. Again, revisit the more detailed discussion of missing data provided in the Modeling Process Section.

Training and Testing

The notebook shows how to use scikit-learn* to split the data into a training set for building the model and then a testing set to evaluate the model's performance. This step is essential to ensure that your model will generalize to new data. The notebook also demonstrates how to perform K-Fold cross-validation.

In the example of K-Fold cross-validation with K=5, the data is split into a training and a testing partition of 80 percent and 20 percent, respectively. A model is trained and then evaluated on the test partition using a specified evaluation metric. This is repeated K number of times, such that by the final iteration it is possible to determine how variable the evaluation metric was between different realizations of the model. If the evaluation metric has low-variability then you can have confidence that the final model will generalize well to new data. On the other hand, if the model performance is highly variable between iterations then try using a less complex model.

Comparing Model Performance

For binary classification problems, the Area Under the Receiver Operating Characteristic Curve, roc_auc_score, is a quick way to compare different classifiers against each other. To truly understand the performance of a classifier, a detailed analysis of the confusion matrix for that classifier is required.

The confusion matrix makes it possible to calculate and compare many different binary classification metrics for your test data. In general, the closer the number of false positives and false negatives are to zero, the better the classifier is. However, in practice you will have to make a trade-off between the two, requiring a thoughtful analysis and consideration of operational factors. For example, if the decision is made to ensure the false negative rate is close to zero (i.e., predict as many RRT events in advance as possible), the hospital staff must be willing to handle an excessive number of "false alarms" (false positives). This may result in a condition called alarm fatigue, in which hospital staff become desensitized to alerts and alarms.

Hyper Parameter Tuning

Most models have multiple hyper parameters that can have significant impacts on how well a given model performs. To ensure the best performing model, grid search – in conjunction with K-Fold cross validation – is a useful way to select the best hyper parameters for a model.

Below is an example of how to use Python's scikit-learn library to conduct a grid search with cross validation.

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import roc_auc_score
from sklearn.grid_search import GridSearchCV

# Specify a grid of possible hyperparameter values.
paramGrid = {'n_estimators': [100, 200, 300],
             'learning_rate': [0.1, 0.05, 0.15],
             'max_depth': [3, 4, 5, 6],
             'min_samples_leaf': [1, 2],
             'subsample': [0.75, 1.0, 0.85],
             'loss': ['deviance'],
             'max_features': [None, 'auto']
            }

# Create a GridSearchCV object for K=5 folds and an evaluation metric of 'roc_auc'.
gs = GridSearchCV(GradientBoostingClassifier(),
                  param_grid=paramGrid,
                  scoring='roc_auc',
                  n_jobs=-1,
                  cv=5,
                  verbose=10)

gs.fit(X_train, y_train)

# Look at the cross validation results
print gs.cv_results_

# Create a final model using the best parameters from the grid search.
finalModel = GradientBoostingClassifier(**gs.best_params_)

# Retrain the production model on all available data.
productionReadyModel = finalModel.fit(X, y)
```

Solutions Proven by Your Peers

The model for predicting RRT events, powered by Intel® technology, provides advanced notice of patient conditions that are likely to result in RRT events. This and other solutions are based on real-world experience gathered from customers who have successfully tested, piloted, and/or deployed the solutions in specific use cases. The solutions architects and technology experts for this solution implementation guide include:

- Andrew Bartley, Senior Solution Architect, Intel
- Michael Hood, Data Scientist, Analytics Group, ProKarma
- Dennis Read, Analytics Architect, Analytics Group, ProKarma
- Emily Spahn, Data Scientist, Analytics Group, ProKarma

Intel Solution Architects are technology experts who work with the world's largest and most successful companies to design business solutions that solve pressing business challenges. These solutions are based on real-world experience gathered from customers who have successfully tested, piloted, and/or deployed these solutions in specific business use cases.

Find the solution that is right for your organization. Contact your Intel representative or visit **intel.com/healthcare**.

Learn More

You may also find the following resources useful:

- Project Site at 01.org
- GitHub Repository
- Intel Predictive Clinical Analytics Business Brief
- Intel Predictive Clinical Analytics Solution Brief
- Intel Datacenter Products
- Cloudera
- Cerner

Solution Provided By: