SUPERVISED MACHINE LEARNING AT PROMINENCE

PROMINENCE

TABLE OF CONTENTS

Introduction 3
Framing the Problem 4
Collecting Data 5
  The Dataset 5
  Feature generation 6
  Relevant tools and capabilities 6
Model Development 7
  Model configuration 7
  Feature importance 9
  Simulation 10
  Relevant tools and capabilities 10
Model Deployment 11
  User testing, silent deployment, and pilot 11
  Implementation 12
  Evaluation 12
  Relevant tools and capabilities 13
Conclusion 14

Introduction

At Prominence, we help healthcare organizations do more with their data, and we strive to make healthcare smarter. Predictive analytics – including data science, machine learning, and generative AI – achieves this transformation when integrated with healthcare providers and experts. Prominence works hand-in-hand with healthcare organizations to lay the foundation necessary to leverage predictive analytics effectively.

This paper focuses specifically on Prominence's approach to supervised machine learning, a key aspect of a strong Data Science practice. To define some terms: machine learning is the process of building an algorithm that learns patterns from historical data, applies those patterns to new data via prediction, and refines its understanding continuously as new data is created – all without explicit programming for those patterns. And supervised refers to algorithms trained to predict a particular outcome variable, unlike unsupervised tasks that seek to identify patterns without a predefined target.

For patients, a well-implemented predictive model can lead to earlier interventions, better health outcomes, and an enhanced experience. And from a hospital administration perspective, such a model could reduce long-term costs, conserve scarce resources, improve hospital rankings, aid in strategic planning, and boost satisfaction for both patients and staff.

"Machine learning is the process of building an algorithm that learns patterns from historical data, applies those patterns to new data via prediction, and refines its understanding continuously as new data is created – all without explicit programming for those patterns."

I will illustrate Prominence's predictive analytics methodology and best practices using a recurring example: a model predicting the likelihood of readmission within 30 days for newly discharged diabetic patients. I'll go through the process of framing the problem, collecting data, and developing and deploying a trained model. A note to readers: throughout the paper, I will include snippets of code to illustrate the modeling process. Feel free to skip over these if you are so inclined; it should not detract from your understanding.

Framing the Problem

Many machine learning projects fail at the outset because they jump to a specific machine learning tool or algorithm too soon. Focusing too much on the type of predictive model to build can result in failing to address the actual opportunity at hand – of missing the forest for the trees. A more successful strategy is to start with the specific healthcare problem you aim to solve – especially understanding why it needs solving.

In general, a good starting point is to enumerate the decision points within the process at hand. Machine learning can be particularly impactful when applied to frequently made decisions. The premise is that consistently accurate predictions can incrementally improve decision outcomes, leading to substantial long-term benefits.

Strategically identifying where a prediction can exert leverage will help clarify the tactical machine learning approach. For instance, do providers need to select from multiple discrete options for a patient? Use classification. Do analysts need to detect when a metric deviates from its historical norm? Use anomaly detection. Do administrators need to plan based on expected future patient population levels? Use forecasting.

In our example, the target solution is reducing the rate of 30-day readmissions for diabetic patients. There are several reasons to pursue this goal – it will produce better patient health outcomes, it will reduce patient and hospital cost, and it will help a hospital's quality metrics. As hospital administrators, we need to identify which patients may require intervention post-discharge to prevent readmission and determine the appropriate interventions.

"Strategically identifying where a prediction can exert leverage will help clarify the tactical machine learning approach."

To improve this decision, I could build a model to predict the probability that each discharged diabetic patient will be readmitted within 30 days. With this information, I can: a) rank patients based on their need for assistance from limited hospital resources; and b) assess whether the number of at-risk patients exceeds available resources to aid in future staffing decisions. Furthermore, understanding the features that lead to a high readmission probability allows us to tailor interventions for individual high-risk patients.

Lastly, I need to determine the timing of the prediction, which will be essential in the next section (*"Collecting Data"*). In this case, I'm going to make the prediction upon discharge so that I can implement post-discharge interventions.

A brief note on tooling: I used a Databricks notebook and cluster to build the model, primarily using various Python libraries (e.g. pandas, scikit-learn).

Collecting Data

The Dataset

Supervised machine learning algorithms need historical data consisting of an outcome to predict, or “target variable”, and numerous related data points, or “features”, from which the model can learn the patterns that predict future outcomes. Features can come from structured data (e.g. database tables) or unstructured data (e.g. natural language text or audio), but their form and timing must comply with the time horizon at which a prediction will be made. In other words, a data point that is not available in a specific format at the time a future prediction is needed is not eligible to be included in the model. To use our recurring example: given that this is historical data, we may know *why* a patient was readmitted, but it would be “cheating” to use that in the model training, as it won’t be available when making real-world predictions.

Identifying relevant datasets and merging multiple data sources accurately involves close collaboration between data scientists and domain experts, such as SQL DBAs and healthcare SMEs. Since a model’s performance hinges on the quality of data it’s trained on, this step is critical and demands strategic handling.

For our diabetic readmission dataset, the target variable is a binary classification: whether or not a patient was readmitted within 30 days of discharge. I have approximately 80,000 unique encounters, paired with a reasonably wide range of features, including length of stay, number of diagnoses, discharge dispositions, and some basic demographic data. Altogether, I have about 50 total features to feed into the model. Note that this is a de-identified, publicly available dataset, so it’s missing other valuable data categories I would ideally include, such as patient history, clinical notes, insurance details, and more comprehensive demographic information. (See table below for a selection of data from our readmission training set.)

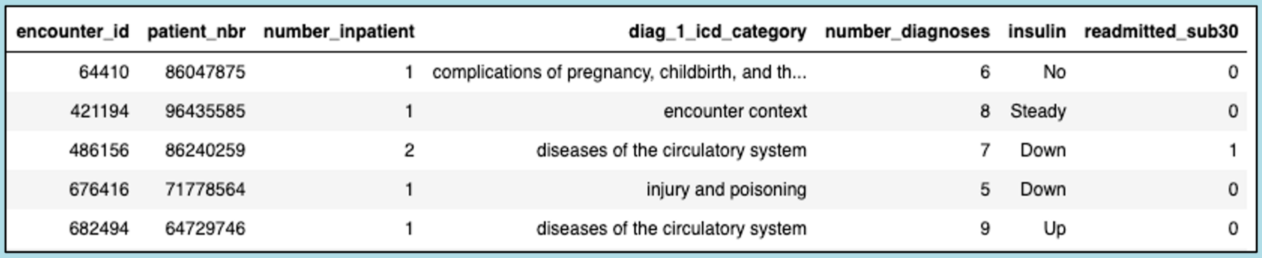
It's important to remember that this model predicts readmission upon discharge. If the scope were to change to predict readmission while the patient is still hospitalized, I would need to create a different dataset containing only the information available prior to discharge.

At a high and slightly obvious level, machine learning models require high-quality data to perform optimally. Ideal datasets are both “deep” (i.e., many instances of the outcome to predict) and “wide” (i.e., a number of data points beyond easy human analysis capacity). Such datasets are more likely to facilitate pattern recognition and are generally beyond the scope of manual human analysis.

“Identifying relevant datasets and merging multiple data sources accurately involves close collaboration between data scientists and domain experts. Since a model’s performance hinges on the quality of data it’s trained on, this step is critical and demands strategic handling.”

More nuanced, though, is the fact that *more* datasets and variables should be included, rather than less. Machine learning algorithms can sift through dozens or hundreds of variables to identify which are relevant for predicting an outcome, so it’s unnecessary at best and detrimental at worst to trim the data the model trains on, at least at first. There could be performance reasons to do so later in the model deployment process, but at first, an unbiased approach to data inclusion is generally preferable.

Training data: ID columns, selected features, and target variable



encounter_id	patient_nbr	number_inpatient	diag_1_icd_category	number_diagnoses	insulin	readmitted_sub30
64410	86047875	1	complications of pregnancy, childbirth, and th...	6	No	0
421194	96435585	1	encounter context	8	Steady	0
486156	86240259	2	diseases of the circulatory system	7	Down	1
676416	71778564	1	injury and poisoning	5	Down	0
682494	64729746	1	diseases of the circulatory system	9	Up	0

Feature generation

Where human intuition *does* become critically important is during a process called “feature generation.” In this stage of the machine learning workflow, a data scientist works with one or more SMEs to create novel data points to enhance the model’s learning capabilities. This helps the model identify significant patterns that aren’t explicitly encoded in the dataset. Some common examples: calculating BMI using height and weight, categorizing pre-existing conditions as chronic or not, and flagging medications known to have interaction effects.

For feature generation in our readmission model example, I grouped diagnosis codes into ICD categories and flagged whether they included a letter or decimal, which signifies different levels of specificity or types of categorization.

In summary, a machine learning model needs a dataset with as many historically representative feature-outcome pairs as is practical. This enables the model to learn relevant patterns to effectively predict the outcome in question when faced with new combinations of features in real-world scenarios.

Relevant tools and capabilities

- Querying data, using SQL
- Data cleaning and preparation, using (typically) Python or R
- IDE, in this case Databricks but could also be VS Code, Jupyter Notebooks, Azure ML Studio, Snowflake, or RStudio
- Feature generation, incorporating Healthcare SMEs

Model Development

Now that I have a dataset, the next step is to fit a model to my historical data. The first step is crucial – I split my data into training and test sets. (See code snippet below.) I'll experiment with various model configurations *only on the former*; I keep the test set out of the initial training as a simulation of a future scenarios where the model must accurately predict outcomes for new data. Holding out a test set helps prevent "overfitting," where the model learns only the contours of the training dataset rather than generalized patterns that can be applied to new data.

```python
from sklearn.model_selection import train_test_split

# Sample data and target variable
X = use_df.drop(target_col, axis=1)
y = use_df[target_col]

# Define the columns to be used in the model
X_model = X.drop(cols_to_ignore, axis=1)

# Store the original IDs
original_ids = X[ID_cols]

# Split data into train and test sets
X_train, X_test, y_train, y_test, ids_train, ids_test = train_test_split(X_model,
y, original_ids, test_size=0.25, random_state=42)
```

Next, the data needs to be cleaned and prepared for the algorithms I plan to test. This is a critical, and often time-consuming, part of the machine learning workflow. Without diving into too many details, a data scientist must handle missing, uncommon, and outlier values, and properly encode features for the machine learning algorithms. Without this preparation, powerful and publicly available algorithms will not function.

Model configuration

To find the optimal model configuration to maximize predictive ability, a data scientist designs an experiment to build and evaluate multiple candidate models. This involves testing:

- various algorithms suitable for the problem type (e.g. RandomForest, XGBoost, ARIMA)
- hyperparameters for each algorithm (e.g., maximum depth for a Random Forest, learning rate for XGBoost)
- the number of features included in training

For our readmission model, I divided the historical patient encounter data into training and test sets using an 80/20 split. I then set up a machine learning experiment using Hyperopt (an open-source Python library for hyperparameter optimization) and MLflow (an open-source platform for managing machine learning workflows) to determine the combination of model, hyperparameter, and feature selection that offers the best generalized predictive performance.

Specifically, I tested RandomForest, XGBoost, and LightGBM classifiers along with a range of their respective hyperparameters. (Feature selection is included in hyperparameter testing for these algorithms.) I used cross-validation – further splitting the training data into smaller subsets while exploring the hyperparameter space – to minimize the risk of overfitting. (See snippet below for calling the optimization code.)

```
# Running the optimization, logging in MLflow
spark_trials = SparkTrials(parallelism=4)   # multi-threaded
best = fmin(fn=objective_function,
            space=space,
            algo=tpe.suggest,
            max_evals=100,
            timeout=3600,
            trials=spark_trials

print("Best: ", best)
```

There are a few different methods for evaluating the model at this stage of development. First, and most objectively, a data scientist chooses a specific evaluation metric to compare different candidates. For our readmission classification model, I chose the model with the lowest log loss as the best model configuration. Log loss is ideal for classification tasks with probability outputs because it accounts for model confidence and handles imbalanced datasets well. In this case, my top-performing model was LightGBM, which I used with my best hyperparameters to train a final model on the entire training set.

To confirm I didn't overfit my model, I used the final model to predict the likelihood of readmission in less than 30 days on both the training and test sets and calculated the log loss for each. The training set log loss was .316, and the test set log loss was .314. The minor difference, with the test set actually performing slightly better, confirmed that I passed this checkpoint successfully.

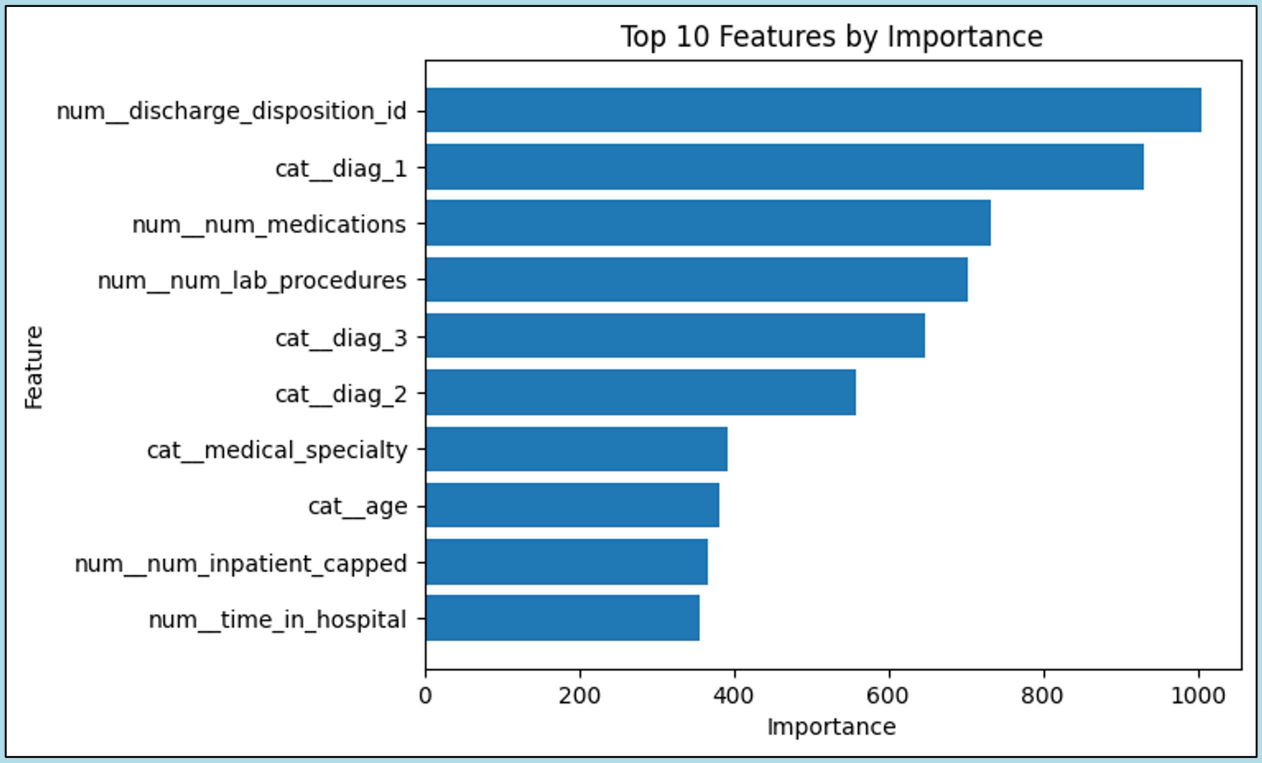
Feature importance

Part of evaluating a model is understanding its “reasoning.” Tools such as SHAP values, permutation importance, and coefficient values help explore the relative importance of different features to the model. While their technical details are beyond this paper's scope, these tools provide valuable insights into what factors influence the model’s decisions, thereby building trust in its accuracy and applicability. Exploring feature importance can reveal non-obvious insights about the real-life context, which is beneficial. However, if the results seem too surprising, further investigation might be needed to ensure there are no errors in data preparation or modeling.

For our readmission model, the top features seem plausible; Discharge Disposition ID, a patient’s diagnoses medications, and the number of days inpatient all make sense as predictors! (See below for a table of top features; note that the prefixes “num_” and “cat_” refer to a given feature’s data type – numeric or categorical.)

“These tools provide valuable insights into what factors influence the model’s decisions, thereby building trust in its accuracy and applicability.”

Feature Importance chart, final model



Simulation

The final evaluation tool we'll examine is simulation, where I create a historical counterfactual analysis. This technique attempts to re-create what would have happened had the model been available historically. Comparing these simulated outcomes to actual historical results allows us to assess the model's performance relative to a hospital's existing process, known as BAU ("business as usual").

For our readmission model, given that this is a de-identified public dataset, I don't have a definitive BAU to compare to, but I can use a simplified hypothetical process to illustrate. Let's assume the hospital currently ranks readmission probability based on length of stay and admission severity (defined by admission source). In this hypothetical scenario, the hospital predicts readmission likelihood using the average readmission rates for each combination of length of stay and admission source.

Comparing my top model's log loss to the hypothetical BAU's, I find that the machine learning model outperforms BAU .314 to .336 - not a dramatic improvement, but it's better. (See code snippet below.) It also highlights the importance of basing decisions on real-life KPIs and outcomes rather than hyper-specific evaluation metrics. That seemingly small difference could have nuances that make it significantly more effective at reducing readmission rates, or it might indicate that I have more work to do to refine the model.

```
# ML vs. sim log loss
print("Log Loss on Test Set, Predicted:", log_loss(y_test, y_pred_proba))
print("Log Loss on Test Set, Simulated:", log_loss(y_test, y_sim_pred_proba))
```

Leveraging multiple evaluation tools is crucial for objectively improving model quality, building stakeholder trust in its predictions, and determining when it's time to promote the model from development to pilot / production.

Relevant tools and capabilities

- ML development, using Python libraries such as pandas, scikit-learn, hyperopt, and MLflow
- Feature importance, using Python libraries such as scikit-learn, xgboost, lightgbm, and SHAP
- Simulation, incorporating Healthcare SMEs

Model Deployment

After validating that a model is producing accurate and meaningful predictions using quantitative evaluation metrics and simulations, I next need to implement the model in production (i.e. use it in real clinical or operational workflows.)

User testing, silent deployment, and pilot

There are a number of considerations when deploying a model, starting with user testing. Early feedback on how accurate, both objectively and subjectively, the model’s predictions are when confronted with new, real data is an important source of information for effectively implementing the model. This can often take the form of a prototype pilot involving a select group of trusted end users.

With the caveat that our readmission model is a hypothetical exercise given the nature of our dataset, I can still illustrate how I would deploy a readmission propensity model for diabetic patients being discharged. Our specific deployment goal is to stack-rank discharged patients to prioritize interventions to prevent near-term readmission.

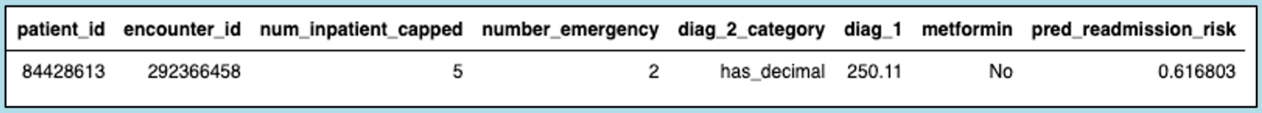
“Early feedback on how accurate, both objectively and subjectively, the model’s predictions are when confronted with new, real data is an important source of information for effectively implementing the model.”

In the initial deployment stage, I would deploy the predictions silently for all newly discharged diabetic patients to monitor real-world predictive accuracy and begin user testing with members of the intervention team. The goal of the silent deployment is to ensure that the model has successfully generalized from the training data to real-world scenarios.

For the user testing stage, I’d randomly select a subset of discharged patients and present their readmission predictions along with the most influential features to our SMEs – clinicians and intervention team members. (See table below for an example of an individual feature importance chart with an accompanying prediction.) I’m seeking subjective feedback on the plausibility of the predictions and features, as well as objective data on the interventions’ efficacy at lowering readmission rates.

By leveraging this feedback, I refine the model as needed before a full-scale deployment, ensuring the model is both accurate and trusted by end users. This process increases the likelihood of the model's success in practical applications and helps build stakeholder confidence in its predictions.

Top 5 features with predicted readmission rate, selected patient



patient_id	encounter_id	num_inpatient_capped	number_emergency	diag_2_category	diag_1	metformin	pred_readmission_risk
84428613	292366458	5	2	has_decimal	250.11	No	0.616803

Implementation

Once the initial pilot deployment is complete, a full rollout involves integrating the model's predictions into existing end user workflows (e.g. a patient's EMR), setting up a cadence for the model to continuously learn and update over time, and scoring new instances as available.

The actual implementation into end user workflows varies on a case-by-case basis, but general best practice is to embed the prediction into the main tools end users already utilize. For the intervention team in our hypothetical readmission model deployment, if they are primarily checking patient EMRs, we'd integrate predictions there, but if they're just relying on a ranked dashboard, we could create our own. This dashboard could also be annotated with detailed insights explaining why patients are ranked in a particular order.

One of the strengths of machine learning models is their ability to learn and adapt over time. We can set up a process for regularly adding new data to the original training dataset, such as nightly or weekly updates. During this process, I will automatically test model types, hyperparameters, and feature selections, similar to the initial development phase, and programmatically promote the top performer to production. With our readmission model, this means I could dynamically switch between models, such as from logistic regression to RandomForest, based on performance data from new instances with no new coding needed. This is especially important if our interventions are as successful as we'd hope in changing the dynamics of readmission!

New prediction generation, called "scoring" or "inference," also needs to be engineered to occur automatically. This can be achieved either via real-time scoring, in which every new instance's prediction is calculated and published to the end user workflow individually and as quickly as possible, or batch scoring, where I pull the latest round of un-scored instances and infer predictions in bulk.

To illustrate the difference, imagine two different processes for our readmissions model. In a real-time scoring scenario, discharge paperwork could trigger the model to update the readmission risk score dynamically, whereas in a batch scoring paradigm, the model would pull all patients discharged the previous day and infer their readmission risk scores en masse. Either way, I am providing a unique and personalized prediction for every patient, but the timeliness of the risk score differs. Note that there is a tradeoff between timeliness and complexity; it is trickier to implement a real-time scoring system than a batch scoring one, so it's important to analyze the cost-benefit ratio when deciding between the two options.

Evaluation

At this stage of the modeling process, evaluation shifts from historical counterfactuals to monitoring and measuring new, live data. The pilot process I mentioned earlier is an important evaluation mechanism and, where practical, can include or expand to a true split test, where I compare the new machine learning-enhanced workflow against BAU and evaluate based on the true KPI for the project. Remember that the goal of the model is not just accurate predictions, but improving key decisions and outcomes!

For our readmission model, this would mean randomly assigning discharged patients into a treatment group that would receive the model's predictions or a control group that would get

BAU probabilities. By comparing the readmission rates of both groups over time, we can determine if the model results in a statistically significant reduction in readmissions.

To reiterate, it's crucial to judge the model on the target KPI and not just predictive accuracy. Circling back to the very beginning of the machine learning workflow, we need to focus on the problem we're trying to solve – readmission rate reduction – rather than the model's predictive power in a vacuum. A highly accurate model that can't reduce readmission rates due to poor implementation is less valuable than a moderately accurate model that effectively augments and improves hospital processes. Accurately measuring ROI from a machine learning project will always mean tracking the performance improvement it drove in the targeted KPI(s).

Surfacing insights into individual predictions is also a worthwhile evaluation exercise at this stage, as it will increase interpretability and, therefore, trust in the model's predictions. There are many tools (e.g. SHAP values, LIME) designed to highlight the specific features that are contributing most significantly to a given prediction. For example, if a patient has a high predicted readmission risk, I could show that they were prescribed >5 medications, and that the model has identified that as a strong predictor of readmission likelihood.

"It's crucial to judge the model on the target KPI and not just predictive accuracy. A highly accurate model that can't reduce readmission rates due to poor implementation is less valuable than a moderately accurate model that effectively augments and improves hospital processes."

Lastly, we need ongoing monitoring of the model's performance. While the model can adapt over time, its performance can deteriorate due to changes in the underlying data or new edge cases. Automated monitoring should alert stakeholders and data scientists if there are issues such as model builds failing, missing inferences for new instances, significant deviations in average predictions from historical data, or worsening model evaluation metrics. While end users are an important source for finding bugs, having as much automation as possible will go a long way toward finding them quickly.

Relevant tools and capabilities

- Data visualization for experiment tracking, model insights, and dashboard implementation if necessary), using Tableau, PowerBI, or the like, incorporating healthcare SMEs
- EMR technical experts (if implementing in patient EMRs), incorporating healthcare SMEs
- ML engineering, in this case using Databricks, but could also be Snowflake, Azure ML services, AWS Sagemaker, or GCP
- Data monitoring, using general purpose tools like Tableau or PowerBI, or purpose-built tools like DataDog or Grafana

Conclusion

When applied to the right decision point with effective interventions and predictive-analytics best practices, machine learning is a powerful tool to make healthcare smarter and to do more with your data. This collaborative effort involves healthcare providers, SMEs, administrators, data scientists, and engineers, and it can create value across clinical, operational, financial, pharmaceutical, and personnel domains. Below is a non-exhaustive list of potential machine learning solutions. These examples might be directly relevant for your organization or serve to illustrate the types of problems where machine learning is most likely to be effective.

If you are interested in tackling a specific problem using predictive analytics or want to learn more about how Prominence can help you build a robust Data Science and Machine Learning foundation at your organization, please get in touch:

Andrew Williams
Director of Data Science, Prominence Advisors
andrew.williams@prominenceadvisors.com

Machine Learning Solutions: a selected, non-exhaustive list

Revenue Cycle	**Clinical**	**Operations / Personnel**
DRG classifier	Fall risk	Provider attrition risk
No-show appointments risk	Hospital readmission	Staff overtime likelihood
Hospital length of stay		
Denials reduction		
House census		