Model Factory: A New Way to Look at Data Through Models

by

Yonglin Wu

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

© Massachusetts Institute of Technology 2016. All rights reserved.
Model Factory: A New Way to Look at Data Through Models

by

Yonglin Wu

Submitted to the Department of Electrical Engineering and Computer Science on May 18, 2016, in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, we present Model Factory, a software framework that is able to generate predictive models from raw data. We present two foundational representations for data: an event-driven time series and a feature series. Together, they allow us to define a large suite of predictive modeling problems, and to subsequently solve them.

We applied Model Factory to two real world datasets: one made up of sensor recordings from prototype cars, and the other containing time-varying status values for projects managed by a consulting firm. We deployed Model Factory on each of these datasets. Through the framework, we were able to enumerate a total of 3,877,848 predictive problems for the car dataset and 125,028 for the project dataset. We randomly sampled 150 and 1,000 prediction problems from the two datasets respectively, and solved them using off-the-shelf machine learning algorithms. We demonstrated our ability to build models for these prediction problems, and to gain insights into the data. We also built a graphical user interface on top of Model Factory for less tech-savvy users.

Thesis Supervisor: Kalyan Veeramachaneni
Title: Principal Research Scientist
Acknowledgments

I would like to thank many people for their help, support and guidance, this thesis would not be possible without them.

First, I would like to thank my advisor Kalyan Veeramachaneni for his continuous support and mentorship. Kalyan is an excellent advisor that truly cares about his students. His feedback has not only helped my research, but also helped me as a researcher and student.

Second, I'd like to thank my sponsors for their support on the project. This thesis would not be possible without their data and financial support.

I’d also like to thank my labmates in ALFA group, especially Skyler Sato and Ben Schreck, who did a lot of work on translating raw data into a schematized format. I was able to use the software they built for my work in this thesis and they were patient and helpful in that process. It was a pleasure working with them.

Lastly, I’d like to thank my parents and my fiancée Siyi for their love and unconditional support all these years.
### 3.2.2 Limitation of the language

36

### 3.2.3 Model output

37

### 4 Experiments and Results

39

#### 4.1 Overview of two datasets

39

#### 4.1.1 Car dataset

39

#### 4.1.2 Project dataset

40

#### 4.1.3 Number of prediction problems

41

#### 4.2 Generate random prediction problems

41

#### 4.3 Model results

42

##### 4.3.1 Comparison of model results of two datasets

42

##### 4.3.2 Individual prediction problem analysis

48

### 5 Graphical User Interface

55

#### 5.1 Selecting channels

55

#### 5.2 Setting up the prediction problem

56

#### 5.3 Histogram

56

#### 5.4 Model output

58

### 6 Conclusion

61

#### 6.1 Key findings

61

#### 6.2 Contributions

62

### A Converting car dataset to event-driven time series

63

#### A.1 Raw data

63

#### A.2 From raw data to event-driven time series

65
List of Figures

2-1 Two views of feature series data. The first view is a matrix with four columns, another view is a matrix with all features on the same row, a flattened version of the first view ............................................................. 24
2-2 A third view of the feature series: slices of feature series data ................. 24
2-3 Apply a feature function on event-driven time series of an entity instance. We apply the feature function on data in the first segment to get the first feature value, apply the feature function on data in the first two segments to get the second feature value, etc. All feature values are saved in a list. ............. 26
3-1 How to slice the data into two parts along a time index. One part is used as input to machine learning algorithm, the other part is used to generate output label for the prediction problem. ............................................. 32
3-2 Select all features of an entity instance from all features in feature series .... 32
3-3 From all features of the same entity, select only features of a specific feature index ................................................................. 33
3-4 Flatten feature series so that features of all time indices are on the same row 34
4-1 Histogram of mean of AUC of prediction problems in project dataset ....... 43
4-2 Histogram of mean of AUC of prediction problems in car dataset ........... 43
4-3 Histogram of standard deviation of AUC of prediction problems in project dataset ................................................................. 44
4-4 Histogram of standard deviation of AUC of prediction problems in car dataset 44
4-5 Histogram of number of features in prediction problems in project dataset ... 45
4-6 Histogram of number of features in prediction problems in car dataset ....... 46
# List of Tables

2.1 Numerical *feature functions* ................................................. 27
2.2 Categorical *feature functions* ................................................. 27

3.1 Output feature functions ......................................................... 33

4.1 Basic statistics of the car dataset ............................................. 40
4.2 Basic statistics of project dataset ............................................. 40

A.1 *Channels* Table Schema ....................................................... 64
A.2 *Messages* Table Schema ....................................................... 64
A.3 *MessageData001* Table Schema ............................................. 64
A.4 Channels Schema CSV ........................................................... 66
A.5 Trip Schema CSV ................................................................. 66
A.6 Signal CSV ................................................................. 66
Glossary

**entity** An entity is a central concept with independent existence in the data, such as a car, a student, or a project. It is similar to the concept of a “sample” in a traditional machine learning context.

**field** A field is an unique identifier denoting the type of events. In MOOCs (Massive Open Online Courses), fields describe the types of actions recorded on courseware, such as “watch video”, “submit problem”, or “post question on forum”. In a data set with recordings of prototype car sensors, fields are the names of signals or recordings, such as “latitude”, “longitude”, or “minute”. “Field” is a universal concept and should appear in all events to ensure context and meaning.

**field type** A “field type” denotes the data types of a field, such as text, image or numbers. There are two field types in Model Factory, “numerical” and “categorical.” “Numerical” fields have continuous values, such as latitude and longitude. “Categorical” fields use integer values to represent categories. For example, in the car recording data set, “engine on” is a categorical field, where 0 means “on” and 1 means “off”.

**timestamp** A “timestamp” is a label representing the time at which the event occurred. It can be a timestring, such as (4/26/2016, 9:35:53 PM), or the seconds elapsed since a designated time. The specific format and unit of the timestamp does not matter, as long as it can be converted into a number with certain units.
Chapter 1

Introduction

1.1 The data science pipeline

We are living in a world where data is generated everywhere all the time. Every button we click on a website, every search we make with Google, and every step we take with our smartphones in our pockets is recorded, stored and analyzed. With this abundance of data in hand, data scientists are constantly asking themselves how they can make the most of it. A common use for this data is in building predictive systems: by analyzing and finding patterns of events in the past, we can intelligently guess what will happen in the future, and take actions accordingly.

However, building predictive systems is not an easy task. Generally, it involves collaboration between domain experts, data scientists, engineers and sometimes designers. To derive insights and build powerful prediction systems, data science teams usually go through the following steps.

First, engineers must extract and aggregate schematized data from a raw data source. The data source can come in many different formats, such as an SQL database, logs on a website, or simply structured text files. Usually, this raw data is not ready to be used directly, and must be aggregated from multiple sources, cleaned, and formatted. Since data formats are usually drastically different in different platforms, this part of the data science pipeline is hard to optimize, and often impossible to build generic frameworks for.

Next, data scientists need to define the prediction problems. In some data sets, the
prediction problem is straightforward. For example, for the development team of an online shopping website, an obvious problem is predicting whether the user will complete on order. The team can study which events are predictive of an order being completed, and use that information to optimize the website design. For other data sets, the prediction problem is not obvious, and the output label is generated from values in the data set. For example, a car data set may contain recordings of the car’s latitude. An interesting thing to predict might be the mean of these latitude recordings for the entire trip, which can be generated from the the data set.

After defining the prediction problem, data scientists must go through a step called **feature engineering**. This is the process of computing complex features from raw data. It’s an important step, because the quality of the features has a huge effect on the predictive power of the model, sometimes even more than the machine learning algorithms. If the features used to train machine learning algorithms have no effect at all on the output label of a sample, we won’t be able to get good results, no matter how good the machine learning algorithms are. As an example, in car recording data set, a feature that may prove useful in predicting the end latitude of a car is “Number of times the engine has started in the first 50 percent of the trip”.

The last step in this pipeline is **machine learning**. After generating features from raw data and storing them in a representation called a “feature matrix” (where the columns are the value of the features and the rows are the samples), data scientists can use standard machine learning techniques to build models and prediction systems. (There are abundant libraries and frameworks developed by previous researchers for this purpose– some of the most notable ones include scikit-learn, Spark and libSVM.)

### 1.2 Motivation for model factory

Model factory is built to automate steps in the data science pipeline. Every step in this pipeline, except for the work of extracting data from raw data sources, can be automated. Plenty of research has focused on automating each step, as does a lot of open-source software. However, in order to complete the whole pipeline, data scientists must use multiple tools,
and spend a lot of time piecing all the steps together.

Model Factory provides a novel approach to tackling this problem. After the data is extracted from a raw data source, Model Factory automates all three subsequent steps: feature engineering, defining the prediction problem, and machine learning. It provides an end-to-end model-building solution for data scientists— all they need to do is convert raw data into a general, standard representation, and then Model Factory will generate models for a large set of pre-specified prediction problems. Data scientists use a specific language, covered further in a later chapter, to define the prediction problem, and are able to get the model for that prediction problem in minutes. We believe Model Factory will drastically improve the efficiency of data scientists, allowing them to spend time thinking about how to use the results of predictive problems rather than spending time designing and solving them.

1.3 Thesis outline

This thesis is organized as follows:

- Chapter 2 presents the two types of data representations in Model Factory, the input data event-driven time series and intermediate data feature series.

- Chapter 3 explains how prediction problems are set up in Model Factory.

- Chapter 4 first describes two real world data sets that Model Factory is used for, and then analyzes the results of experiments we ran on those two data sets.

- Chapter 5 shows a graphical user interface we built on top of Model Factory for less technical users.

- Chapter 6 discusses the the key findings from Model Factory, and its contributions to data science community.
Chapter 2

Data Representations and Abstractions

Model Factory is designed for generating predictive models on a particular type of dataset: event-driven time series data. Event-driven time series data contain events, each of which has a timestamp, a field and some value associated with it. This type of dataset has the following characteristics:

1. **Event Driven.** Recording values for a field happens only when an event happens. For example, there is a dataset that has recordings from sensors installed on a prototype car, which we refer to as “car dataset”. In this dataset, there is a field “Door Status Change”. It has 2 values, 0 and 1. 0 means the door closes and 1 means door open. Recording of this field only occurs when the door opens or closes. When that happens, a recording of value 0 or 1 with “Door Status Change” field and timestamp of that time is recorded in the database.

2. **Irregular.** Events happen at irregular time intervals. Using the same examples as above, the event “Door Status Change” may occur once in the first hour, never occurs in the next three hours, and happens twice in the fifth hour. There is no regular interval of when the events occur. This type of data is different from physiological data, in which signals such as blood pressure and heart rate are continuously monitored and recorded on a regular basis.

3. **Not all fields have recordings for each entity.** For example, in the same “car dataset”, there is a field called “Light Activation” that has value of 1 when light turns
on and 0 when light turns off. If the light was never turned on, in a car trip there are no recordings with field name “Light Activation”.

2.1 Event-driven time series representation

Event-driven time series is a collection of data entries, each with the following 5 attributes. It is the standard input data representation for Model Factory.

- **entity_id**: Entity_id is an unique identifier for Entity. Entity is a central concept with independent existence in the data, which can be a car, a student, or a project. It is similar to the concept of “sample” in traditional machine learning context.

- **timestamp**: Timestamp is a label that represents the time which the event occurred. It can be a timestring, such as (4/26/2016, 9:35:53 PM), or number of seconds since a designated time. The specific format and unit of the timestamp does not matter, as long as it can be converted to a number with certain units.

- **field_id**: Field_id is an unique identifier for a field while field is an unique identifier of the type of events. In MOOC (Massive Open Online Course), fields are type of actions recorded on courseware, such as “watch video”, “submit problem” or “posting question on forum”. In a dataset with recordings of sensors on prototype cars, fields are names of the signals or recordings, such as “latitude”, “longitude”, “Minute”. It is a universal concept and should appear in all events, otherwise the meaning of the events are missing.

- **value**: Value is the value for the field.

- **field_type**: Field type is the data type of a particular field, such as text, image or numbers. There are two field types in Model Factory, “numerical” and “categorical”. “Numerical” fields are fields that have continuous values, such as latitude and longitude. “Categorical” fields are the fields that use integer values to represent categories. For example, in car recording dataset, “engine on” is a categorical field. In that field, 0 means “on” and 1 means “off”.

20
Example 1: Massive Open Online Courses
An example of a dataset that can be converted into event-driven time series is data collected from MOOC (Massive Open Online Course) websites. For each student using MOOC such as edX (https://www.edx.org/), the course website records all activities of every student, such as submitting homework, watching lecture videos and posting questions on a discussion forum. [3]

To convert this dataset into event-driven time series, each student needs to be assigned with a unique entity id. Each type of event, such as “video watched”, “questions answered” needs to be assigned with a unique field id.

Fields that have continuous values are assigned with field type numerical and fields that have discrete values are assigned with field type categorical. For example, suppose “video watched” event stores the number of seconds the students has watched the video. Since the number of seconds is a continuous value, “video watched” field has field type “numerical”. “Problem check success” recording, on the other hand, has value of 1 if the check is successful, and 0 if it fails. Since it only has two values, “problem check success” field is a categorical field. In MOOC dataset, each recording also has a time string with the format like “4/26/2016, 9:35:53 PM”. This time string is stored in timestamp column directly.

Example 2: Car dataset
Another example of such dataset is a collection of signals generated from prototype cars a company is developing. We will refer this as car dataset later in this chapter. There are thousands of different signals, including common ones such as latitude and longitude, and also recordings from sensors specifically designed for the car that only domain experts understand.[2] Each entity in this dataset is a trip taken by a car. Each type of signal such as “latitude” and “longitude” are assigned a unique “field id”.

<table>
<thead>
<tr>
<th>entity id</th>
<th>field id</th>
<th>value</th>
<th>field type</th>
<th>timestamp</th>
</tr>
</thead>
</table>
In this dataset, there is an attribute “signal type” associated with each type of signal, which has six possible values: “Analog”, “Digital”, “State Encoded”, “Hybrid State Encoded”, “Text” and “Double”. When converting the data into event-driven time series, signals with signal type “Analog”, “Digital” and “Double” are assigned with field type “numerical”. Signals with signal type “State Encoded” and “Hybrid State Encoded” are assigned with field type “categorical” and signals with signal type “Text” are assigned with filed type “Text”. Each signal also has a timestamp whose value is the number of seconds since Jan 1 1970. This number is stored directly in “timestamp” column in even-driven time series.

2.2 Feature series representation

The main premise of Model Factory is that it can build models on the fly with just the input data. In order to build complex and good models, we often need to produce features from raw data. Features are statistics or numerical combination of values from a single or multiple channels over a period of time. The purpose of building “features” from raw data is to provide more complex signals that makes it easier for machine learning algorithms to train. However, building features from raw data is a time consuming process and takes too much time to do for every model. To save time, Model Factory pre-processes the input data and produces an intermediate data representation which we call feature series and stores it on disk. Once that’s done, model factory can load feature series into memory and operate on the in-memory feature series data and skip the work of generating features from raw data. But then the question is how to generate a generic “feature series” representation so it can be used across predictive problems. Next, we will define the representation of feature series.

Feature Series is a collection of all feature values generated from raw data. Each entry in the feature series representation has the following attributes:

- time index

  Time index is an identifier for when the feature is calculated. Raw data is divided into
segments of equal time interval and features are computed at the end of each segment. Time index is an integer between 1 and the number of segments. Features computed and the end of $nth$ segment has feature index $n$.

- **feature index**

  Feature index is an unique identifier for the features. It is generated when features are computed from raw data.

- **entity id**

  Features are computed from a set of entries in event-driven time series that have the same entity id. Thus each feature has a single entity id associated with it.

- **feature value**

  This is the value of the feature. It is usually the result of the feature function applied on a list of raw input values.

2.2.1 **Multiple views of feature series**

Feature Series can be viewed in multiple ways, as listed below.

- The first way it can be viewed is the way it is stored on disk. It is a matrix with four columns: entity id, time index, feature id and value.

- Another view of feature series is a matrix with all the features of a certain entity at a certain time index on the same row. It’s essentially what we get if we take the first matrix and flatten all features of the same time index of an entity into the same row. It is a view that data scientists and machine learning engineers are more familiar with, because in traditional machine learning context, all features of the same entity are on the same row.

- A third view of the feature series is slices of feature matrices in which each feature matrix contains all features of all entities at a certain time index. Each row of the feature matrix are all features of the same entity. It is a view that is more suitable
Figure 2-1: Two views of feature series data. The first view is a matrix with four columns, another view is a matrix with all features on the same row, a flattened version of the first view.

Figure 2-2: A third view of the feature series: slices of feature series data for the context of Model Factory. In Model Factory, since we are using event-driven time series data with a timeline, it is natural to use data from the past to predict some value in the future. When doing this kind of predictions, we will cut the feature series into two parts. Some slices will be used as input and some will be used to generate output label. This will be covered in details later.
2.2.2 Converting event-driven time series into feature series

- **Step 1**: Extract a time series corresponding to a field and an entity

  When computing features, the first step is to extract out that time series with only the recordings from a single entity and field. When extracting time series, we sort the recordings by timestamp for easy segmentation later on. For example, in the car dataset, a time series can be all recordings of field *Latitude* on trip 88.

- **Step 2**: Check the field type

  Two types of fields are supported in Model Factory, **categorical** and **numerical**. For example, in the car dataset, *Latitude*, which is the recording of the current latitude, is a numerical field because it is a continuous value. “Field Type” is specified in event-driven time series and the work of determine the field type is left for user who convert raw data into event-driven time series.

- **Step 3**: Break the time series into segments

  Model factory uses some data in the past to predict some values in the future, so features need to be computed at different times in the time series. Since computing features at every possible time point takes too much time, model factory only computes features at certain times. To do this, we first break the time series into equal sized segments. The number of segments can be specified by the users.

  For example, in the car dataset, a time series can be all recordings of field *Latitude* on trip 88. The total span of the trip is 5 hours. After sorting the recordings by timestamp, we break the time series into 5 segments, where each segment last one hour. The first segment contains all recordings in the first hour, the second segment contains all recordings in the second hour, etc.

- **Step 4**: Apply feature functions cumulatively

  There are two types of feature functions, numerical and categorical. The type of field (categorical or numerical) is determined in step 2, and all feature functions of this
Figure 2-3: Apply a feature function on event-driven time series of an entity instance. We apply the feature function on data in the first segment to get the first feature value, apply the feature function on data in the first two segments to get the second feature value, etc. All feature values are saved in a list.

The type will be applied on the time series. We use a cumulative strategy to calculate the features for each feature function. To calculate the first feature, we use all data from first segment. To calculate the second feature, we use all data from the first two segments, etc. Features calculated from the first $N$ segments have time index of $N$. All features values calculated for this time series are saved in a list.
<table>
<thead>
<tr>
<th>Feature Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>latest_val</td>
<td>Input value at closest timestamp</td>
</tr>
<tr>
<td>sum</td>
<td>sum of all previous values</td>
</tr>
<tr>
<td>sum_abs</td>
<td>sum of absolute value of all previous values</td>
</tr>
<tr>
<td>mean</td>
<td>average of all previous values</td>
</tr>
<tr>
<td>mean_abs</td>
<td>average of absolute value of all previous values</td>
</tr>
<tr>
<td>max</td>
<td>maximum of all previous values</td>
</tr>
<tr>
<td>max_abs</td>
<td>maximum of absolute value of all previous values</td>
</tr>
<tr>
<td>min</td>
<td>minimum of all previous values</td>
</tr>
<tr>
<td>min_abs</td>
<td>minimum of absolute value of all previous values</td>
</tr>
<tr>
<td>var</td>
<td>variance of all previous values</td>
</tr>
<tr>
<td>last_minus_first</td>
<td>Current Value minus first value</td>
</tr>
<tr>
<td>maxdiff</td>
<td>Maximum of value minus previous value(diff)</td>
</tr>
<tr>
<td>mindiff</td>
<td>Minimum of value minus previous value(diff)</td>
</tr>
<tr>
<td>absmaxdiff</td>
<td>Maximum of absolute value of value minus previous value(diff)</td>
</tr>
<tr>
<td>absmeandiff</td>
<td>Average of absolute value of value minus previous value(diff)</td>
</tr>
</tbody>
</table>

Table 2.1: Numerical feature functions

<table>
<thead>
<tr>
<th>Feature Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Most common value so far</td>
</tr>
<tr>
<td>Jitter</td>
<td>Number of times the value has changed</td>
</tr>
<tr>
<td>Stability</td>
<td>number of times the most common value has been seen divided by the total number of values so far</td>
</tr>
<tr>
<td>Total_X</td>
<td>Number of times the value is X</td>
</tr>
<tr>
<td>Percent_X</td>
<td>Fraction of times the value is X</td>
</tr>
</tbody>
</table>

Table 2.2: Categorical feature functions

For example, we want to calculate all features of the time series of field “latitude” that has 5 segments in a trip of 5 hours. Since “latitude” is a numerical feature, we apply all feature functions in Numerical Feature Functions table. Suppose we are calculating feature “mean”, which is the average of all previous values. The first feature is calculated by taking the average all recordings in the first segment, which has all recordings in the first hour. This feature has **time index** of 1. The second feature is calculated by taking the average all recordings in the first two segments, which have all recordings in the first two hours. We repeat this until we reach the last segment, with which we compute the average of all recordings of “latitude” in the trip. This process is repeated for all other numerical feature functions, such as “latest_val”, “sum”, etc.
To make the example more concrete, let’s assume there is time series of field “speed” with 3 segments, the recordings in each segments are listed as follows:

- Segment1: 10, 20, 30
- Segment2: 30, 60
- Segment3: 50, 10

Suppose we are applying the feature function “mean” on this time series, which computes the average of all previous values. The feature functions need to be applied for time index 1, 2 and 3. The feature values at those time indices are computed as follows:

- Time index 1: feature value = \((10 + 20 + 30) / 3 = 20\)
- Time index 2: feature value = \((10 + 20 + 30 + 30 + 60) / 5 = 30\)
- Time index 3: feature value = \((10 + 20 + 30 + 30 + 60 + 50 + 10) / 7 = 30\)

### 2.2.3 Number of features for each entity

Model Factory generates many features for each entity instance. The number of features is the same for each entity instance, and it is determined by the number of fields, their types, and number of feature functions available for that type. Below we show how the number of features can be calculated using the parameters of number of fields and number of feature functions.

1. At each time index, features are computed for each field. the number of time index is the same as the number of segments, which is specified by the user. We use \(t\) to refer to this number. Since the same number of features are calculated at each time index, the total number of features is the number of features at a given index multiplied by \(t\).

2. Features generated for each field at a given time index is different for numerical and categorical field.
If the field type is “numerical”, the number of features at a given time index equals to \( k \), the number of numerical feature functions.

If the field type is “categorical”, there are two types of categorical features:

(a) Categorical feature functions that operate on the whole field and generate one feature for the field, which we refer to as static categorical feature functions. Examples of these type of feature functions are “mode”, “jitter”, and “stability”. There is only one “mode” for each field. Suppose the number of those type of feature functions is \( m \), the total number of features generated by them is \( m \).

(b) Categorical feature functions that operates on each categorical value of this field, which we refer to as dynamic categorical feature functions. Example of those functions are “Total X”, “Percent X”, where “X” is a specific value the categorical variable takes. For example, if a categorical field has possible value of 0,1,2, then there are 3 features for “Total X”, which are “Total 0”, “Total 1”, “Total 2”. Suppose the number of those type of feature functions is \( p \), and the number of categorical values is \( c \), then the total number of features generated is \( p * c \).

In summary, the number of features generated for each field is \( k \) if the field is numerical and \( m + p * c \) if the field is categorical. Suppose there are \( n_n \) numerical fields and \( n_c \) categorical fields, the total number of features at a time index is \( n_n * k + n_c * (m + p * c) \).

Thus the total number of features in feature series can be calculated with the following parameters:

- \( t \): number of time index (segments)
- \( k \): number of numerical feature functions
- \( m \): number of static categorical feature functions
- \( p \): number of dynamic categorical feature functions
• $c$: number of categorical values

• $n_n$: number of numerical fields

• $n_c$: number of categorical fields

And the total number of features at all time indices is:

$$(n_n \ast k + n_c \ast (m + p \ast c)) \ast t$$

(2.1)
Chapter 3

Prediction Problem Setup

3.1 Setting up prediction problem

To set up a prediction problem from the precomputed feature series, we follow the following steps.

3.1.1 How to slice the data?

In previous chapter, we showed that one representation of feature series is slices of feature matrices, each slice has all features of all entities are a certain time index $T$. In order to make prediction problem, we need to cut the data into two parts. One part of the data will be used as input for machine learning algorithm, and the other part will be used to generate output labels. The way to cut the data is along a time index $T_0$ that user can specify. All the features slices that have time index smaller or equal to $T_0$ will be use as input for prediction problem; everything else will be used to generate output labels for the prediction problem.

3.1.2 How to generate the output label?

For each entity we need to generate an output label, from the feature slices we have. Here are the steps we need to take to compute the output label from feature slices.

- **Select an entity**: We select the feature values for an entity instance, since the output label of an entity will only be computed using the features of that entity.
Figure 3-1: How to slice the data into two parts along a time index. One part is used as input to machine learning algorithm, the other part is used to generate output label for the prediction problem.

Figure 3-2: Select all features of an entity instance from all features in feature series

- **Select a feature and aggregate the values:** Out of all feature values for one entity, we select feature values of a single feature_id. Only a single feature_id is used because it is uncertain in a generic dataset, whether output label that are computed from multiple features have actual meaning.
Figure 3-3: From all features of the same entity, select only features of a specific feature index

- **Apply an aggregation function:** After the first two steps, now we have a list of feature values from the entity instance and same feature, but different time indices. We will use an **Output Operator Function** to aggregate that list of feature values into a single number. The table below lists all “output feature function” available for Model Factory.

<table>
<thead>
<tr>
<th>Output Feature Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>Maximum of the list of values</td>
</tr>
<tr>
<td>Min</td>
<td>Minimum of the list of values</td>
</tr>
<tr>
<td>Sum</td>
<td>Sum of the list of values</td>
</tr>
<tr>
<td>Latest_val</td>
<td>Last of the list of values</td>
</tr>
</tbody>
</table>

Table 3.1: Output feature functions

- **Threshold to generate a label:** Lastly, after aggregating the feature values with a output feature function, the output becomes a single numerical value. The prediction problems Model Factory defines have binary values as output label. The binary output label is then computed by comparing the numerical output value from previous step with an **output threshold** that the user specifies. Thus, finally we have an entity_id, label pair.
3.1.3 Flatten feature series for machine learning

In Section 3.1.1, we cut out some slices from feature series to input into machine learning algorithm. However, in order to use in to machine learning algorithm, we need to convert the slices into a 2D feature matrix, in which in row of the matrix contains values of all the features of all time indices. To do that, we need to flatten the feature slices and combine features of each slice into the same row, as shown in Figure 3-4:

![Figure 3-4: Flatten feature series so that features of all time indices are on the same row](image)

3.1.4 Using machine possible learning algorithm

Right now we have the two sets of data we need to apply machine learning algorithm. There is a feature matrix where each row contains all features before a time index of an entity, and different rows are different entity instances. There is also a vector of output labels that contains an binary output label for each entity instances. Those are standard input data for supervised machine learning algorithms, thus we can apply a number of classification algorithms to train the predictive model, such as neural network, naive bayes, or decision tree. In our implementation, we used Random Forest classifier. As standard machine learning practice, we split the data into train and test sections, whose ratio can be specified by the user.
3.1.5 Number of prediction problems

With model factory, we can generate a number of prediction problems. Suppose there are \( t \) time indices, then there are \( t - 1 \) ways to cut the data, because we can cut it at time index 1, 2, ... \( t - 1 \). We cannot cut the data at time index \( t \) since if we do that there won’t be any data to generate the output label.

Suppose there are \( f \) features at each time index, \( l \) output feature functions and \( s \) output thresholds, the total number of prediction problem Model Factory can generate is:

\[
(t - 1) \times f \times l \times s
\]  

(3.1)

3.2 Model factory language and result

We have demonstrated that a prediction problem can be set up by specifying four parameters: feature index, time index, output feature function and output threshold. From previous chapter, we can see features are computed by applying feature functions on a specific field, so each feature index can be uniquely defined by field_id and a feature function.

3.2.1 Model factory language

Now that we can show the prediction problem can be specified by 5 parameters:

- field_id
- feature function
- time index
- output feature function
- output threshold

The five parameters form a language to define a prediction problems in Model Factory. The goal of Model factory is to define this language as general as possible so that it can cover
a lot of different prediction problems. From last chapter, the total number of features at
each time index can be calculated as follows:

\[ f = n_n \ast k + n_c \ast (m + p \ast c) \]  \hspace{1cm} (3.2)

Thus the total number of possible prediction problem is:

\[ (t - 1) \ast f \ast l = (n_n \ast k + n_c \ast (m + p \ast c)) \ast (t - 1) \ast l \ast s \]  \hspace{1cm} (3.3)

### 3.2.2 Limitation of the language

Even though the language is very flexible, we cannot setup any arbitrary prediction problems
with it. The total number of possible prediction problem is still a finite number. There are
a few main limiting factors in how the prediction problems can be set up:

- **Feature function and output feature function**

  The type of prediction problems we can setup is largely determined by those two sets of
  functions. The limitation for the user is that they have to choose from the set of feature
  functions that are already implemented in Model Factory. We supply as many functions
  we can in Model Factory and user can add their own, but that still can not cover all
  possible feature functions user may want to use. Also, not all prediction problem can
  expressed with combination of two functions; some are complex and require a lot more
  functions.

- **Time series setup**

  They way model factory is setup is that user can use some data in earlier part of
  the entity to predict some data in later part of the entity. The setup is also a main
  limitation of Model Factory, as it removes possibility to all prediction problems that are
  not time-related, such as using values of some fields to predict values of another field
  at any given time. Even for time-related prediction problem, Model Factory doesn’t
  cover all scenarios since it only provides one cut in time axis for user. If the user, for
example, wants to predict some values between 50 percent to 80 percent on the time
series using data between 20 to 40 percent, it cannot be done with Model Factory.

3.2.3 Model output

Model Factory builds a model for each prediction problem. It presents two sets of information
that is useful to users: **top 10 features** and **AUC score**.

- **Top 10 features**
  
  Model Factory uses many features to build a model for predicting output label, though
  not all features are weighed equally in the model. After building the model, Model
  Factory then presents users 10 features that have the most weights in the model. Those
  are the most predictive features for the current prediction problem. Data Scientists can
  then use those features to build their own models. Even if the features are not directly
  usable, looking at the features can help data scientists discover which fields are highly
  correlated with the output label. For each feature, **field name**, **feature function**,
  **percentage** and **weight** are presented to users. Weight represents the importance
  of the feature in the predictive model. Important features have large weights in the
  model.

- **AUC score**

  AUC, which stands for Area Under the Curve of ROC, is a common evaluation metric
  for binary classification problems. ROC, which stands for Receiver Operating Char-
  acteristic curve, is a plot of true positive rate vs false positive rate as the threshold
  value for classifying an item as 0 or is increased from 0 to 1.[1] AUC is the area under
  ROC. If classifier is very good, ARC should be close to 1. If the classifier is no better
  than random guessing, AUC will be close to 0.5. Model Factory calculate AUC for
every model it builds to give users a sense of how good the model is for this prediction
problem. It helps user decide whether this is a prediction problem worth looking into.

To prevents overfitting, when building models on a dataset, Model Factory performs
a K-fold cross-validation. It splits the data into K equal parts, trains model on K-1 parts and use that model on the part of data that’s left for prediction. An AUC score is calculated for that prediction. Model factory repeats this process K times and computes the mean and standard deviation of the AUCs. In our implementation of Model Factory, we use $K = 10$. 
Chapter 4

Experiments and Results

In this chapter, we demonstrate that we can use Model Factory on real world dataset and retrieve meaningful results. Specifically, we use Model Factory on two datasets, one from prototype cars and one from project managed by a consulting company. We will refer to the dataset from a car company as the **car dataset**, and dataset from a consulting company as **project dataset**.

We first give an overview of those two datasets and then demonstrate how we generate random prediction problems for those two datasets. After that we present the model results we get by running Model Factory on those randomly generated prediction problems. In the end, we analyze a few prediction problems and show how Model Factory can help users gain critical insights of the data.

4.1 Overview of two datasets

4.1.1 Car dataset

The car dataset is a collection of recordings from different sensors installed on prototype cars. There are many different type of sensors, such as accelerometers, gyroscopes, heat sensors, light sensors and each sensor collects multiple type of signals. Recordings are generated when a sensor detects a certain event, such as *temperature change*, *door open* or *user turning on front light*. Each recording contains a signal value, type of the event and timestamp. This
car company has these prototype cars in different cities. When the cars are driven, all the recordings from the sensors are collected during that time. Each continuous run of the car is also referred to as a trip. Recordings from each trip are saved in a separate SQLite database file while the car dataset is a collection of those SQLite files.

The details of how to convert the car dataset into event-driven time series is covered in Appendix A.

### 4.1.2 Project dataset

The project dataset is a collection of time-varying status values of projects that this consulting firm is responsible for. During the span of the project, values of the project status are recorded and saved whenever there is an update on the project. Typical statuses include “total cost”, “total backlog at time start”, etc. In this dataset, each “entity” is a project and each “field” is an attribute. Some attributes have text values, for example, attribute “comments” records the comments of the manager on the project. Since model factory only work with numerical and categorical fields, those attributes are ignored when data is converted to event-driven time series. All data collected from all projects are saved in a MySQL database.

The work of converting project dataset into event-driven time series is done by another member of ALFA lab, Benjamin Schreck.
4.1.3 Number of prediction problems

How many prediction problems can we generate for those two datasets? In Chapter 3, we show that the number is

\[(t - 1) \times f \times l \times s\]  \hspace{1cm} (4.1)

where \(t\) is the number of time indices, \(f\) is the number of features, \(l\) is the number of output feature functions and \(s\) is the number of output threshold. In our implementation of Model Factory, there are 10 time indices and 4 output feature functions. Since we only use median as the output threshold, \(s = 1\).

There are 3473 features in project dataset, so the total number of prediction problem is:

\[(10 - 1) \times 3473 \times 4 = 125,028\]  \hspace{1cm} (4.2)

There are 107718 features in car dataset, so the total number of prediction problem is:

\[(10 - 1) \times 107718 \times 4 = 3,877,848\]  \hspace{1cm} (4.3)

4.2 Generate random prediction problems

To evaluate the performance of Model Factory, we use it to generate models for randomly generated prediction problems and analyze the results. To speed up the process of running prediction problems, we wrote python scripts to repeatedly generate prediction problems, use Model Factory’s python user interface to get model results, and store the model results in a text file.

In the Chapter 3 we presented the language Model Factory used to define a prediction problem. It consists of five parameters user can specify: **field id**, **feature function**, **time index**, **output feature function** and **output threshold**. We performed the following steps to generate those five parameters:

1. Select a random time index between 1 and the maximum time index.
2. Select a random field_id from the list of all possible field_ids in the dataset.

3. If the generated field is a numerical field, select a numerical feature function at random. Otherwise, select a categorical feature function at random.

4. Select randomly an output feature function.

5. Use “getstats” function to get the median of the output value, use that as the output threshold. This is to ensure that are roughly equal number of positive and negative output labels for this prediction problem.

4.3 Model results

We ran 1,000 prediction problems on project dataset and 150 prediction problems on car dataset. First we will compare some key stats of the model for those two datasets, and then we will pick two prediction problems for each dataset, one with excellent AUC score and one with mediocre AUC score and interpret the model results.

4.3.1 Comparison of model results of two datasets

Mean and standard deviation of AUC score

As shown in figure 4-1 and figure 4-2, comparing to project dataset, AUC scores of prediction problems on car dataset have larger mean and smaller standard deviation, which suggests the models are consistently better on car dataset. However, this can also be a result of overfitting, as there are a lot more features in car dataset (107718) than the number of entities (7081). On the other hand, in project dataset the number of features (3473) is smaller than the number of entities (11088).
Figure 4-1: Histogram of mean of AUC of prediction problems in project dataset

Figure 4-2: Histogram of mean of AUC of prediction problems in car dataset
Figure 4-3: Histogram of standard deviation of AUC of prediction problems in project dataset

Figure 4-4: Histogram of standard deviation of AUC of prediction problems in car dataset
Features used for prediction

The number of features is proportional to the time_index for the same dataset. As shown in Figure 4-5 and Figure 4-6, there are only 9 possible feature numbers, one for each time index. The number of prediction problems with the 9 feature numbers are roughly the same, with some variation due to the fact time index is randomly generated. There are a lot more features in car dataset because there are more fields.

Number of entities for training

Not all entities are viable for all prediction problems. If no recordings of the field with the field_id exist in the time series of an entity, the output label cannot be computed. We can see in both car dataset and project dataset, there are many prediction problems where only a small subset of entities can be used to train models.

Ratio of positive output labels

In Model Factory, output label is a binary value. When generating random prediction problems we try to have equal number of positive and negative output labels. Since we use “median” as output threshold and a label is only positive if it is strictly larger then the
Figure 4-6: Histogram of number of features in prediction problems in car dataset

Figure 4-7: Histogram of number of usable entities for prediction problems in project dataset
Figure 4-8: Histogram of number of usable entities for prediction problems in car dataset

Figure 4-9: Histogram of ratio of positive output labels in project dataset prediction problems
Figure 4-10: Histogram of ratio of positive output labels in car dataset prediction problems

output threshold, the ratio of positive label over all the labels is always smaller than 0.5. We can see in both car and project dataset, the majority of the problems have a close to 0.5 positive label ratio.

4.3.2 Individual prediction problem analysis

In this section, for both project and car dataset, we will examine one prediction problem that returns good AUC score and one prediction problem that returns bad AUC score and analyze those results.

Project prediction problem 1

- Prediction problem parameters:

<table>
<thead>
<tr>
<th>Field name</th>
<th>total actual effort assembly test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field feature function</td>
<td>sum</td>
</tr>
<tr>
<td>Output feature function</td>
<td>lastest value</td>
</tr>
<tr>
<td>Threshold</td>
<td>0</td>
</tr>
<tr>
<td>Time index</td>
<td>2</td>
</tr>
</tbody>
</table>

- Model result:
- AUC mean: 0.643
- AUC std: 0.06
- top 10 features:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Feature function</th>
<th>Time index</th>
</tr>
</thead>
<tbody>
<tr>
<td>completed</td>
<td>total_1</td>
<td>2</td>
</tr>
<tr>
<td>previous overall status</td>
<td>total_1</td>
<td>2</td>
</tr>
<tr>
<td>project estimate at completion</td>
<td>max_abs</td>
<td>2</td>
</tr>
<tr>
<td>infrastructure</td>
<td>percent_3</td>
<td>2</td>
</tr>
<tr>
<td>actual headcount</td>
<td>sum_abs</td>
<td>2</td>
</tr>
<tr>
<td>over under stuffing</td>
<td>sum_abs</td>
<td>2</td>
</tr>
<tr>
<td>project variance at completion</td>
<td>var</td>
<td>2</td>
</tr>
<tr>
<td>project estimate at completion</td>
<td>latest_val</td>
<td>2</td>
</tr>
<tr>
<td>schedule variance</td>
<td>min_abs</td>
<td>2</td>
</tr>
<tr>
<td>cost performance index</td>
<td>max</td>
<td>1</td>
</tr>
</tbody>
</table>

**Analysis:**

In this prediction problem, we are trying to predict the sum of the field “total actual effort assembly test” at the end of the project using all data from the first 20 percent of the project. We can see no field appear twice, thus no field is highly correlated with what we want to predict. Low AUC score of 0.643 indicates that it is a difficult prediction problem. Part of the reason is that we’re using the data from first 20 percent of the project to predict a value at the end of the project, which can be difficult.

**Project prediction problem 2**

**Prediction problem parameters:**
### Field name | Reporting Frequency
---|---
Field feature function | Stability
Output feature function | Average
Threshold | 0
Time index | 9

**Result:**

- AUC mean: 0.953
- AUC std: 0.01
- Top 10 features:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Feature function</th>
<th>Time index</th>
</tr>
</thead>
<tbody>
<tr>
<td>previous overall status</td>
<td>percent _3</td>
<td>8</td>
</tr>
<tr>
<td>dominant financial geographic unit</td>
<td>mode</td>
<td>1</td>
</tr>
<tr>
<td>previous overall status</td>
<td>stability</td>
<td>9</td>
</tr>
<tr>
<td>previous overall status</td>
<td>percent _3</td>
<td>7</td>
</tr>
<tr>
<td>dominant financial service group</td>
<td>stability</td>
<td>9</td>
</tr>
<tr>
<td>dominant financial service group</td>
<td>jitter</td>
<td>9</td>
</tr>
<tr>
<td>per earned</td>
<td>min_diff</td>
<td>5</td>
</tr>
<tr>
<td>quality comments</td>
<td>topic _6</td>
<td>6</td>
</tr>
<tr>
<td>per earned</td>
<td>sum_abs</td>
<td>7</td>
</tr>
<tr>
<td>baseline duration</td>
<td>max</td>
<td>2</td>
</tr>
</tbody>
</table>

**Analysis:**

In this prediction problem, we are trying to predict the “stability” (ratio of the number of appearance of most common value divided by total number of values) at the end of the project, using all data from the first 90 percent of the project. The high AUC score indicates that the model does a good job of predicting. From the top 10 features, three fields stand out: “previous overall status”, “dominant financial service group” and “per earned”.
Car prediction problem 1

- **Prediction problem parameters:**
  
<table>
<thead>
<tr>
<th>Field name</th>
<th>PwrSup2AliveCounter_PT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field feature function</td>
<td>sum</td>
</tr>
<tr>
<td>Output feature function</td>
<td>sum</td>
</tr>
<tr>
<td>Threshold</td>
<td>9390.0</td>
</tr>
<tr>
<td>Time index</td>
<td>5</td>
</tr>
</tbody>
</table>

- **Result:**
  
  - AUC mean: 0.9944
  - AUC std: 0.006
  - top 10 features:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Feature function</th>
<th>Time index</th>
</tr>
</thead>
<tbody>
<tr>
<td>VehicleInfoParametersB0 B0 b6</td>
<td>sum</td>
<td>4</td>
</tr>
<tr>
<td>VehicleInfoParametersC0 C0 b5</td>
<td>sum</td>
<td>3</td>
</tr>
<tr>
<td>UB DeactivatedPKeyID2 BO</td>
<td>sub_abs</td>
<td>5</td>
</tr>
<tr>
<td>UB DeactivatedPKeyID1 B0</td>
<td>sum_abs</td>
<td>2</td>
</tr>
<tr>
<td>MinuteCounter Pt b2</td>
<td>var</td>
<td>4</td>
</tr>
<tr>
<td>VehicleConfParameters MS1 B0 b3</td>
<td>sum_abs</td>
<td>2</td>
</tr>
<tr>
<td>UB DeactivatedPKeyID2 BO</td>
<td>sum</td>
<td>5</td>
</tr>
<tr>
<td>VehicleInfoParametersB0 B0 b0</td>
<td>sub_abs</td>
<td>1</td>
</tr>
<tr>
<td>VehicleConfParameters MS1 B0 b5</td>
<td>min_abs</td>
<td>4</td>
</tr>
<tr>
<td>DoorStatusHS CH</td>
<td>sum</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Analysis:**
  
  In this prediction problem, we are trying to predict the sum of the sum of PwrSup2AliveCounter_PT field at 60, 70, 80, 90 and 100 percent of the trip. It is a bit difficult to interpret what the output label means, but the AUC score of the result is excellent. Again in the model we see a few fields stand out: “VehicleInfoParameters”,

51
“VehicleConfParameters” and “UB DeactivatedPKey”. Those fields are hard to interpret and it shows the strengths of Model Factory. Without Model Factory, we will not be able to build features and models since we don’t understand the meaning of each field.

**Car dataset prediction problem 2**

- **Prediction problem parameters:**

<table>
<thead>
<tr>
<th>Field name</th>
<th>EngineTorqChecksum PT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field feature function</td>
<td>mean</td>
</tr>
<tr>
<td>Output feature function</td>
<td>latest value</td>
</tr>
<tr>
<td>Threshold</td>
<td>127.7</td>
</tr>
<tr>
<td>Time index</td>
<td>7</td>
</tr>
</tbody>
</table>

- **Result:**

  - AUC mean: 0.578
  - AUC std: 0.034

- top 10 features:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Feature function</th>
<th>Time index</th>
</tr>
</thead>
<tbody>
<tr>
<td>PwrTrnTorqAct CH</td>
<td>mean</td>
<td>4</td>
</tr>
<tr>
<td>WheelRotToothCntReR HS2 PT</td>
<td>absmeandiffdiff</td>
<td>5</td>
</tr>
<tr>
<td>EngineSpeed CH</td>
<td>absmeandiff</td>
<td>3</td>
</tr>
<tr>
<td>EngineTorqChecksum PT</td>
<td>mean</td>
<td>6</td>
</tr>
<tr>
<td>Clock RC B0 b0</td>
<td>absmeandiffdiff</td>
<td>4</td>
</tr>
<tr>
<td>GPSHeading Raw</td>
<td>mean</td>
<td>6</td>
</tr>
<tr>
<td>PATSDataControl PT b1</td>
<td>mean</td>
<td>3</td>
</tr>
<tr>
<td>PwrSupChargeMode PT</td>
<td>percent_6</td>
<td>4</td>
</tr>
<tr>
<td>AmbientTemp B0</td>
<td>absmaxdiff</td>
<td>1</td>
</tr>
<tr>
<td>UB HLDTouchState C0</td>
<td>absmeandiff</td>
<td>7</td>
</tr>
</tbody>
</table>
• Analysis:

In this prediction problem, we are trying to predict the mean of field “EngineTorqChecksum PT” at the end of a trip. This prediction problem is difficult as we see only on AUC mean of 0.578. In the model, no field names appear twice. We are starting to see a pattern that in hard prediction problems, we don’t usually see multiple features with the same field name, which indicates that the features are selected somewhat randomly and are not actually predictive. In easy prediction problems, however, there are usually repeated fields in top 10 features, which suggests those fields are high correlated with the output label.
Chapter 5

Graphical User Interface

We implemented our own version of Model Factory as a python package, although in principal it can be implemented in any language. User can import Model Factory package into their python project integrate Model Factory with their own workflow by calling specific functions. We chose to implement Model Factory in python because it is a popular language for data scientists and have good data processing and machine learning libraries such as numpy, scipy, matplotlib and scikit-learn.

On top of the python package, we also built a graphical user interface in the form of a web app for users who are less technical and are more interested in building and exploring models using an out-of-box solution. The framework we used to implement this web app is Flask, a light weight python framework. In this chapter, we will go over this web app step by step and explain each page in detail.

5.1 Selecting channels

This is the first page users see after they log in with username and password. On this page, they can select which channels they want to explore. They can select as many channels as they would like on this page. The channels they select will be available to select on the main page. The main reason purpose for this page is to narrow down the set of channels users are interested in. In some dataset there are thousands of channels are the user may only be interested in building prediction problems for a few channels at a time. On this page
they can select that few channels they are interested in. If later they decide to explore more channels, they can always go back to this page and select different channels.

5.2 Setting up the prediction problem

As shown in Figure 5-2, This is a section for user to set up the prediction problem. There are a few configurable fields in this section, and all of them are explained in detail in Chapter 3. Users will see this section on the left side of the webpage after they select channels in previous step.

5.3 Histogram

After setting up the parameters of the prediction problem except for “output threshold”, User can see the stats of the output label on right side of the webpage. This helps user decide a good “output threshold” for the prediction problem.
Figure 5-2: Setting up Prediction Problem Section

Figure 5-3: Displaying Histogram Section
Sometimes the user may be interested in examine the histogram of a particular range of values to refine the output threshold, thus Model Factory gives users the option to replot histogram in a specific range.

5.4 Model output

When user finish setting up the prediction problem and click “Model”, the server will train the model in the background. After it is done, it will give users the top 10 features of the model as well as the AUC score.
### Top 10 features

<table>
<thead>
<tr>
<th>Channel Name</th>
<th>Operator</th>
<th>At Trip Percentage</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABOFault_HS</td>
<td>stability</td>
<td>10%</td>
<td>0.05520105396463097</td>
</tr>
<tr>
<td>BrakeWarningLamp_HS</td>
<td>total</td>
<td>0%</td>
<td>0.049403163000077174</td>
</tr>
<tr>
<td>Update Count</td>
<td>sum</td>
<td>10%</td>
<td>0.0479803667929091</td>
</tr>
<tr>
<td>BrakePressure_HS</td>
<td>absmean</td>
<td>10%</td>
<td>0.04761012174293276</td>
</tr>
<tr>
<td>EngineSpeed_MS</td>
<td>mindiff</td>
<td>10%</td>
<td>0.04253308128544245</td>
</tr>
<tr>
<td>BrakePressure_HS</td>
<td>sum_abs</td>
<td>10%</td>
<td>0.04244292751755438</td>
</tr>
<tr>
<td>WheelSpeedRel_HS</td>
<td>absmean</td>
<td>10%</td>
<td>0.0406857856409698</td>
</tr>
<tr>
<td>Update Count</td>
<td>sum_abs</td>
<td>10%</td>
<td>0.039659175214767486</td>
</tr>
<tr>
<td>DisplayAccessRequest_MS_b6</td>
<td>max</td>
<td>10%</td>
<td>0.039267893558555859</td>
</tr>
<tr>
<td>VehicleInfoParametersHS_HS_b4</td>
<td>sum</td>
<td>10%</td>
<td>0.0389445573154654</td>
</tr>
</tbody>
</table>

**ROC score (Train/Test Split: 80/20)**

0.910160023483

---

**Figure 5-5: Displaying model output section**
Chapter 6

Conclusion

6.1 Key findings

In this thesis, we have presented Model Factory from end-to-end, and demonstrated that it can be used on two real worlds datasets. Here, we provide several key takeaways for this software.

Model Factory can be applied generally to a variety of datasets. We presented general input representation for Model Factory: event-driven time series and demonstrated that any dataset that can be converted into this representation can be used as input to Model Factory. We also converted two real world datasets, one from a car company and one from a consulting company into this representation and used Model Factory to build predictive models for them.

Model Factory can build models for a large number of prediction problems on the fly. We showed that the user can define prediction problems using Model Factory language, and presented the mathematical function to calculate the number of prediction problems Model Factory can generate. For two real world datasets, the number of prediction problems we can use Model Factory on about 125 thousand and 3.9 million respectively. We also demonstrated that we can speed up the process of building models by pre-processing the data and save intermediate feature series data on disk. We presented the process to generate input data and output labels for machine learning algorithms by operating on feature series data.
Model Factory can be used by both technical and non-technical users. Developers who wish to integrate Model Factory into their project can use Model Factory as a python package. People can also use Model Factory as a web app, which has a user-friendly Graphical User Interface that we walked through in Chapter 5.

6.2 Contributions

In this thesis, we:

1. Designed Model Factory, a software that builds predictive models for a large number of prediction problems on a dataset on the fly.

2. Defined Event Driven Time Series and Feature Series, two data formats used in Model Factory. Event Driven Time Series is the input data format for Model Factory, while Feature Series is the intermediate data format that stores all features computed.

3. Implemented Model Factory and ran it on two real world datasets, demonstrating its generalizability.

4. Implemented a Graphical User Interface on top of Model Factory for users coming from a less technical background.

5. Performed experiments using Model Factory on two real world datasets and gained insights on those two datasets.

We conclude that Model Factory can build models for a wide range of datasets, and is a powerful tool for datasets who want to build many predictive models from raw data rapidly.
Appendix A

Converting car dataset to event-driven time series

A.1 Raw data

The dataset given to us contains recordings of thousands of signals from prototype cars developed by this car company, and comes in in following directory structure.

Data/

|-- Car Id/

|--|-- Trip Date/

|--|--|--SQLite database file for one trip

There are multiple parallel directories (e.g. multiple Car Ids) at each directory level, even though only one is listed in here. The lowest level SQLite database file, contains all data recorded for one trip. The total data size is 1TB and there are a total of 7041 trip files that have non-empty signal data. There are 7931 unique signals from all trips. There are many tables inside each SQLite database file, and for the purpose of this project, 3 sets of tables,
### Channels Table Schema

<table>
<thead>
<tr>
<th>Column Names</th>
<th>Column Names</th>
<th>Column Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message ID</td>
<td>ID</td>
<td>TIME VAL</td>
</tr>
<tr>
<td>Name</td>
<td>Name</td>
<td>Signal 1</td>
</tr>
<tr>
<td>Signal Type</td>
<td>ArbID</td>
<td>Signal 2</td>
</tr>
<tr>
<td>Min Value</td>
<td>SignalCount</td>
<td>Signal 3</td>
</tr>
<tr>
<td>Max Value</td>
<td>DataTableName</td>
<td></td>
</tr>
<tr>
<td>Average Value</td>
<td>Network</td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>Min Time</td>
<td></td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>Max Time</td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>Min Delta</td>
<td></td>
</tr>
<tr>
<td>Units</td>
<td>Max Delta</td>
<td></td>
</tr>
<tr>
<td>DAQ Rate</td>
<td>Message Delta</td>
<td></td>
</tr>
<tr>
<td>Decimals</td>
<td>Message Count</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Message Changes</td>
<td></td>
</tr>
</tbody>
</table>

### Messages Table Schema

<table>
<thead>
<tr>
<th>Column Names</th>
<th>Column Names</th>
<th>Column Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME VAL</td>
<td>Signal 1</td>
<td></td>
</tr>
<tr>
<td>Signal 2</td>
<td>Signal 3</td>
<td></td>
</tr>
</tbody>
</table>

### MessageData001 Table Schema

Channels, Messages, and MessageData are of particular interest to us, and their schema are listed here:

A Channel is a type of signal. “Channels” table is a metadata table that stores information about all type of signals collected on a trip. The field “Signal Type” specifies the type of signal it collects. There are 5 types of signal; State Encoded, Double, Analog, Digital and Text.

Signal data are bundle in “Messages”. A message contains signals that are related or collected together. Raw signal data are stored in “MessageData” tables, such as “MessageData001”, “MessageData002”, “MessageData003”. Messages is a metadata table that stores meta information about each Message. “MessageID” and “Message Channel” field in “Channels” table stores information about where a type of signal is recorded. For example, if for Channel 7, MessageID = 5 and Message Channel = 10, then in MessageData005 table, Signal 10 contains values for Channel 7.

In “Channels” table, ID is unique but “Name” is not. However, the assumption we make is that if two channels have the same name, then they are the same channel even though
they may have different channel Id. This is the basis we use to align channels from different trips.

A.2 From raw data to event-driven time series

The Raw SQLite databases may be a good choice for storing and transferring data, but they are too complex for quickly extracting data and running experiments. Thus as a first step, we flattened the data by extracting signals and metadata and place them in three sets of CSVs without losing any relevant information. The schema for those three sets of CSVs are listed in Table 4,5 and 6:

There is one Channels-schema CSV file per trip, storing metadata information about channels in that trip. For the purpose of this project, the car or date of the trip does not matter, so in translating Channels-schema CSVs, we flattened the file directory and stored all “Channel-schema” files under one directory (“all-channels”), appended with a unique integer ID for each trip. (e.g Channel-schema-365.csv). All fields except for “discrete and continuous” in this csv are populated by aggregating information of a certain field (e.g minValue) across all Channels in “Channels” table.

We treat each channel as either “discrete” or “continuous”, and the “Decimals” field in Channels table determines it. If “decimals” field has value of -1, we treat the channel as “discrete”, which is encoded as a value of 0 in this schema.

In order to do machine learning across multiple trips, we need to align Channels in different trips. The assumption we make is that channels with the same name are the same channels, even if they have different IDs in different trips. Under this assumption, we scanned through all channels in all the trips, and created a dictionary of one to one mapping from Channel name to unique Channel ID. The dictionary is saved in the format of a JSON file, with the following schema: Channel_name: Channel_UID
Table A.4: Channels Schema CSV

<table>
<thead>
<tr>
<th>Column Names</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>Min Value</td>
<td></td>
</tr>
<tr>
<td>Max Value</td>
<td></td>
</tr>
<tr>
<td>Discrete or Continuous</td>
<td></td>
</tr>
<tr>
<td>Max Channel State Val</td>
<td></td>
</tr>
<tr>
<td>Max Channel State Description</td>
<td></td>
</tr>
<tr>
<td>Units</td>
<td></td>
</tr>
</tbody>
</table>

Table A.5: Trip Schema CSV

<table>
<thead>
<tr>
<th>Column Names</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Trip ID</td>
<td></td>
</tr>
<tr>
<td>Car ID</td>
<td></td>
</tr>
<tr>
<td>Start time</td>
<td></td>
</tr>
<tr>
<td>Stop time</td>
<td></td>
</tr>
</tbody>
</table>

Table A.6: Signal CSV

<table>
<thead>
<tr>
<th>Column Names</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal ID</td>
<td></td>
</tr>
<tr>
<td>Timestamp</td>
<td></td>
</tr>
<tr>
<td>Signal Value</td>
<td></td>
</tr>
<tr>
<td>Start/Stop/Continuous</td>
<td></td>
</tr>
</tbody>
</table>

There are 7391 unique signals in total, the largest signal file is 3.6 GB, while the smallest one less than 1KB.

Trip schema is one csv file that contains information from all trips. For each trip, we scan through all “Messages” table, find the “start time” and “end time” for each message, and update the “start time” and “end time” for the trip. In the end, for each trip we store the earliest timestamp for a signal as “start time” and the latest timestamp for a signal as “end time”.

For each trip, we also generate a “signal” file that stores all signal recordings of a trip in a flat schema shown above. Similar to “channel schema”, we also append unique trip id to the name of signal file. For each record in the csv, “Signal Id” field is the universal Channel_UID we stored in Channel_UID dictionary. For discrete signals, we only store the signal values when it changes, hence, the “start” and “stop” of a signal. For continuous signals, we store the value at every timestamp. In this schema, “start” is encoded by value of 0, “stop” by 1, and “continuous” by 2.
Bibliography

