The Report Committee for Piyush Prakash
Certifies that this is the approved version of the following report:

Predikt: A Simpler Machine Learning Experience

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor: ______________________________
Adnan Aziz

____
Herb Krasner

Predikt: A Simpler Machine Learning Experience

by

Piyush Prakash, B. Tech.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

The University of Texas at Austin

May 2015

Dedication

I would like to dedicate this report to my family, friends and teachers...

My parents, Narendra Kumar Sinha and Sudha Sinha, for all their
love, encouragement, and guidance.

My wife Brototy, and my son Pratyush for the love, support, and incredible patience as I
went through the Master's program while working full-time.

My sisters Dipty and Deepika for their love and support.

To my teachers and classmates at UT who always helped and supported me,
throughout this incredible journey.

Acknowledgements

I would like to acknowledge my supervisor, Dr. Adnan Aziz, for the support and guidance that he provided to make this project possible. I would also like to thank Herb Krasner for reading and providing feedback on this report.

Abstract

Predikt: A Simpler Machine Learning Experience

Piyush Prakash, M.S.E.

The University of Texas at Austin, 2015

Supervisor: Adnan Aziz

With the exponential growth in volume of data being generated by the businesses today, the need for automated systems to facilitate decision making is growing. In many cases, the volume of data has reached the scale where it is not humanly possible to analyze the relevant data using traditional processes, which are dependent on manual intervention, to reach a good decision. The increase in diversity of data sources (internal, external, social, sensors, etc) has added significantly to the challenge of decision making. Predikt presents a simplified approach for the automation of decision making in organizations using Data Mining and Machine Learning techniques with minimal human intervention. Human Intervention, when required, will typically be to accept, reject or override a decision already made by the machine. Predikt front-end as well as back-end is able to adapt to end users unique requirements. Predikt is able to render User Interface dynamically based on User Inputs, while backend is able to do intelligent data processing such as binning (for numeric columns) and multiple model evaluation.

Table of Contents

List of Tables

List of Figures

Chapter 1: Evolution of Big Data

Growth of data in organizations is reaching proportions beyond the ability of commonly used software tools and processes to capture, curate, manage, and process data within a tolerable elapsed time [1,2]. Big data is a constantly moving target, as of 2015 ranging from a few dozen terabytes to many petabytes of data are being generated today. According to a recent study by IBM, every day, we create 2.5 quintillion bytes of data — so much that 90% of the data in the world today has been created in the last two years alone. This data comes from everywhere: sensors used to gather climate information, posts to social media sites, digital pictures and videos, purchase transaction records, and cell phone GPS signals to name a few [4]. Big data is a set of techniques and technologies that require new forms of integration to uncover large hidden values from large datasets that are diverse, complex, and of a massive scale [3].

Companies that are to survive and be successful in this day and age of data abundance are the ones that will learn how to harness this data to drive intelligent decision making. A common denominator among the successful companies of today is data driven decision making. From Amazon to Walmart, from Google to Facebook, the focus is to enable data driven decision making that is not bound by the limitations of human mind (and bandwidth), but is powered by the scalability of the modern infrastructure and computing capacity. These companies have derived significant value by learning to harness Big Data. Predikt aims to bring the same capability and tools to mass market.

1.1 Current State of Data Science

Data Sciences is a critical component to solving the explosive growth of data, but it is also no secret that the volume, variety, velocity and veracity (4 Vs of Big Data) of data is far outpacing the capacity of manual analysis and conventional data sciences tools. On top of it, practice of data science is widely considered a mysterious magical art far beyond the reaches/understanding of common business users. The commonly held perception (and to a great extent reality) regarding Data Sciences is that it requires an in-depth understanding of the mathematics and complex algorithm to be able to derive meaningful value out of it. Although large companies are able to invest into building dedicated teams with this specialized skillset, this current state has become a bottleneck in the wider adoption of Data Sciences, especially by small and medium sized companies. Thus, opening up an opportunity for us to present a framework that relies on automation and simplification as its guiding principles.

Challenges: There are several challenges in the current state:

a. *Needs heavy investment:* Although large companies are able to invest into creating a dedicated team, small and medium sized companies are not able to invest into a dedicated team of Data Scientists, and typically end up relying on traditional gut-driven decision making.
b. *Lack of skilled resources:* Growing realization among companies about the criticality for data-driven decision making has also led to significant deficit in the market between demand and supply of the trained data scientists. It gets even more difficult to find the right skillset if it is a complex domain, and requires data

scientist to also understand the nuances of the business to be effective and successful.

c. *Disconnected Decision Making:* Even if a company has the capacity to invest into building a dedicated team of data scientists, there is almost always a disconnect between this team and the group that is responsible for building/envisioning the product line. For example, the data analysis and modelling team within an online retail business may be perceived as a scientific organization and their feedback not incorporated into deals and offerings being made on their website. The perception may also be because of a previous attempt at using data mining to drive decisions may not have yielded desired results, and thus creating a resistance in the minds of stakeholders to further optimize it. [9]

d. *Real-time Decision Making:* Most of the time, team that is responsible for data mining does not have real-time access to the data. They try to make recommendations based on data that is old (generally by few weeks and sometimes by months) and does not reflect the current market situation. This could be because the systems/databases in many organizations exist in silos and data scientists do not have direct access to that data. They rely on operations or IT to bring that data into a central warehouse to be able to get access to the data. There is also lag in getting access to data, and cleansing/converting it in a format where it can be fed to a machine learning engine. The delay increases if data needs to be pre-processed by a batch job or map-reduce engines like Hadoop, which is performed by IT teams.

1.2 Case For Simplification

For Data Driven Decision making to be a norm and not an exception, it is critical that the end users should be able to learn and apply the underlying concepts behind Data Sciences and Machine Learning even without a background in mathematical modelling and Statistics. The user experience should be seamless and presented to them in a format that is intuitive, and in the language of their domain.

1.3 Case for automation

If companies and users have to start trusting data and the decision being made by algorithms, it is important to reduce the lag time between choices that users make (input to the model) and the re-training of model. Additionally, steps need to be taken to ensure that the system is able to take corrective actions and be able to make the right choices on behalf of the users. For example, the system should be able to evaluate multiple algorithms based on real-time inputs be able to apply the right model to maximize the users ROI and satisfaction. Automation plays a critical role in ensuring this seamless interaction between users and machines, and enhancing user confidence in machine's ability to take decisions. In such a scenario, user's key role becomes that of providing inputs as needed by the algorithm, and oversight of the outcome.

1.4 Case for Flexibility

For a solution to be adopted by the masses, it needs to be flexible to support multiple domains without manual intervention. For example, the user interface and prediction options shown to an Oil & Gas user should be customized for that domain and rendered dynamically based on data being provided by the end user.

1.5 Project Contribution

This project was able to accomplish the following:

a. Establish a vision for simplification for adoption of Machine Learning in organizations that do not have the resources to invest into a dedicated Data Science team and/or tools.
b. Design of an application that meets the requirements derived from the guiding principles – Simplification, Automation and Flexibility.
c. Implementation, QA and Release of **Predikt** - An application that aims to bring the power of Machine Learning and Predictive Analytics to mass market.

1.6 Organization of this Report

The next chapters of this report are organized as follows:

- **Chapter 2 - Vision:** This chapter presents the high level vision for Predikt by laying out the guiding principles, requirements and features.
- **Chapter 3 - Application Design:** This chapter presents the high level design for Predikt including Mockups, User Stories, User Flows and Technical Design.
- **Chapter 4 – Project Results**: This chapter presents various qualitative as well as quantitative results from the project including screenshot and Software Engineering Metrics.
- **Chapter 5 - Conclusion:** This chapter concludes the report by discussing future work, lessons learnt and what went right in the project.

Chapter 2: Vision

2.1 GUIDING PRINCIPLES:

Predikt attempts to solve the challenges discussed in the previous section by following these Guiding Principles:

a. Simplicity
b. Automation
c. Domain Agnostic

Following **requirements** are derived from the above stated guiding principles:

a. A business user with no background in data science should be able to use the application with minimal training.
b. Should be able to support various domains (healthcare, finance, Oil & Gas, etc) with minimal configuration.
c. User Interface should be presented to user in the context/language of their domains.
d. Should enable seamless publishing of latest data into the engine.
e. Model re-training and comparison between various models (to pick the right model given latest data) should be abstracted from the end-users.
f. Should allow end users to override the decisions made by the engine (for example, user can chose to use a different algorithm from the one recommended by Predikt to do their prediction).

2.2 FEATURES:

Deriving from the high level guiding principles and requirements, following is the list of features needed in Predikt. These features can be divided into two categories:

2.2.1 Predikt Responsibility:

These are the set of features that Predikt should be able to accomplish without any input/action from the end users. The output form these features can be made available to end users for review and actions, but the process itself should be automated behind the scene.

- Rendering of UI based on user's domain and Input data format
- Intelligent data pre-processing
    - Example - Binning for Numeric parameters, Handling of outliers/negative/null values, training and test set identification, etc.
- Training of multiple models based on latest data provided by the customer
- Comparison of training outcome and recommendation of the optimal model

2.2.2 Customer Responsibility:

These features require an action or input by the end users. These will need a user interface for end users to perform these actions. These features can be further divided into the end users role and the level of familiarity with Machine Learning models.

i. Basic Features:

These features require no previous data science background and can be performed by a business user.

- **Initial Setup:** This is the one time step to configure Predikt specific to customer's domain/requirements. Predikt team will work with end users to finalize their specific prediction requirements, input data format, relevant models to be evaluated, frequency of data refresh, etc. Predikt will be configured based on the collected requirements.
- **Ongoing Responsibility**: Customer will be responsible for providing latest data to Predikt on an ongoing basis. Predikt will use this data to train and compare various models outcome, which will result in a recommended model for the customer that can be used for prediction (till new input data is provided).

ii. **Advanced Features:** These features require data science background, and should be performed by an advanced/designated user.

- **Override the recommended model:** Users will be able to review various technical parameters related to models, that were retrained based on the latest data, including the recommended model. User will be able to override the Predikt recommended model
- **Configure data pre-processing rules:** Advanced users will be able to override default data pre-processing rules, such as number of bins for numeric columns, how to handle outliers, negative and null values.

Chapter 3: Application Design

Following chapter presents the high level design for Predikt. The sections included in this chapter are:

a. UI Design (Mockups)
b. User Stories
c. User Flow
d. Technical Design

3.1 UI Design

Mockups were prepared in Balsamiq to envision the overall user experience for Predikt.

Mockup 1 shows the overall layout for the Main page. The main page is divided into three main sections:

1. **Model Training -** This section is used to upload new data and setup data pre-processing rules. .
2. **Model Training Status Check -** This section is used to track the status of the model training.
3. **Model Execution -** This section is used to predict an outcome by providing input parameters.

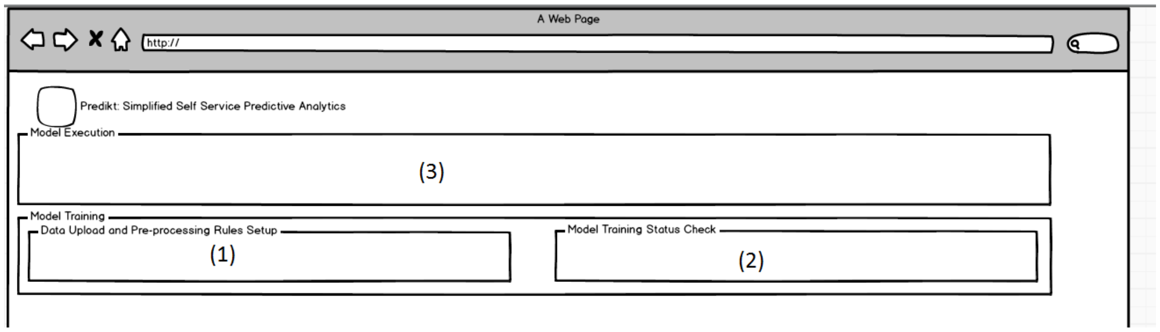
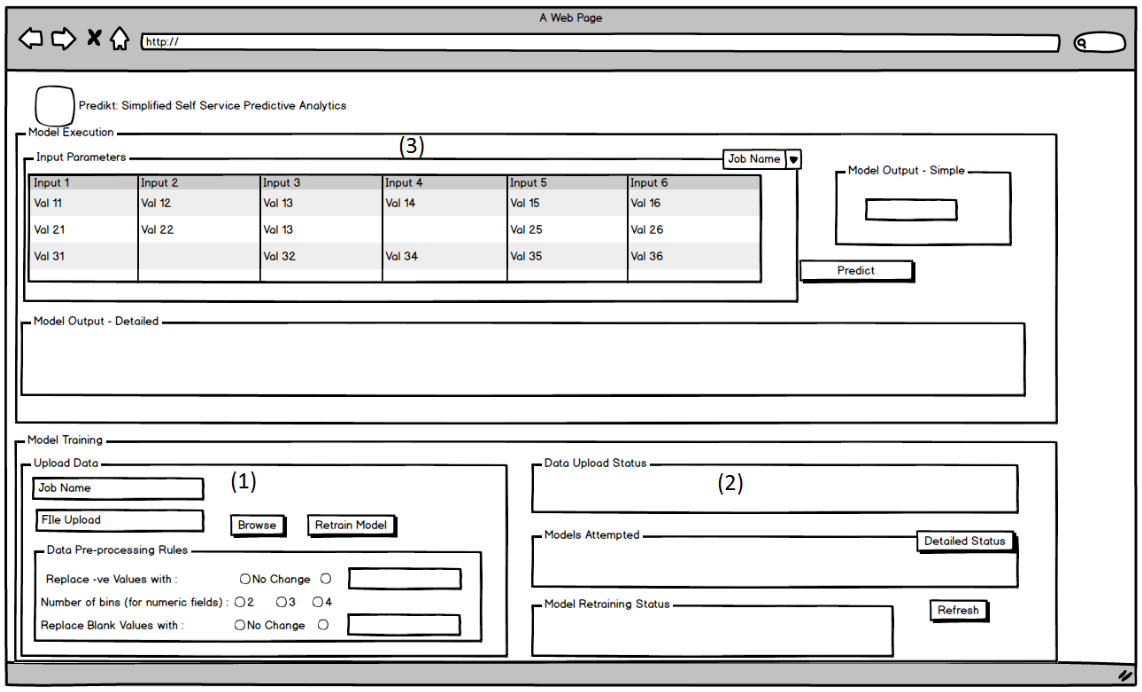

A Web Page
http://
Predikt: Simplified Self Service Predictive Analytics
Model Execution
(3)
Model Training
Data Upload and Pre-processing Rules Setup
(1)
Model Training Status Check
(2)

Figure 1: Main Page Layout Mockup

Mockup 2 shows the detailed Main page with the controls filled into the three sections highlighted in Mockup 1.



Input 1	Input 2	Input 3	Input 4	Input 5	Input 6
Val 11	Val 12	Val 13	Val 14	Val 15	Val 16
Val 21	Val 22	Val 13		Val 25	Val 26
Val 31		Val 32	Val 34	Val 35	Val 36

Figure 2: Main Page Mockup

Design for each of the three sections is discussed below:

3.1.1 Data Upload and Pre-processing Rules Setup

This section is (marked at Section 1 in the Main Page Mockup) users to upload new data, and set data pre-processing rules. A new uploaded dataset and the associated pre-processing rules can be labeled by giving a name (Job Name). Finally, a new model retraining job is kicked-off by clicking the “Retrain Model” button from this section.

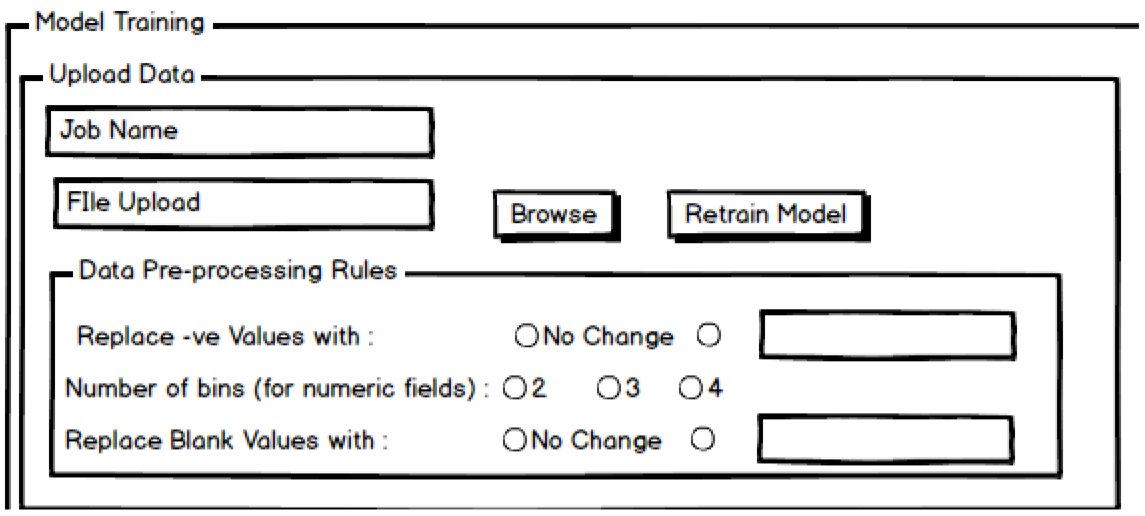

Model Training
Upload Data
Job Name
FIle Upload
Browse
Retrain Model
Data Pre-processing Rules
Replace -ve Values with :
No Change
Number of bins (for numeric fields) :
2
3
4
Replace Blank Values with :
No Change

Figure 3: Section 1 Mockup

Additionally, user is also given an option to pick the columns from the new dataset that they would like to incorporate into the model (as Input Parameters) by providing the screen below:

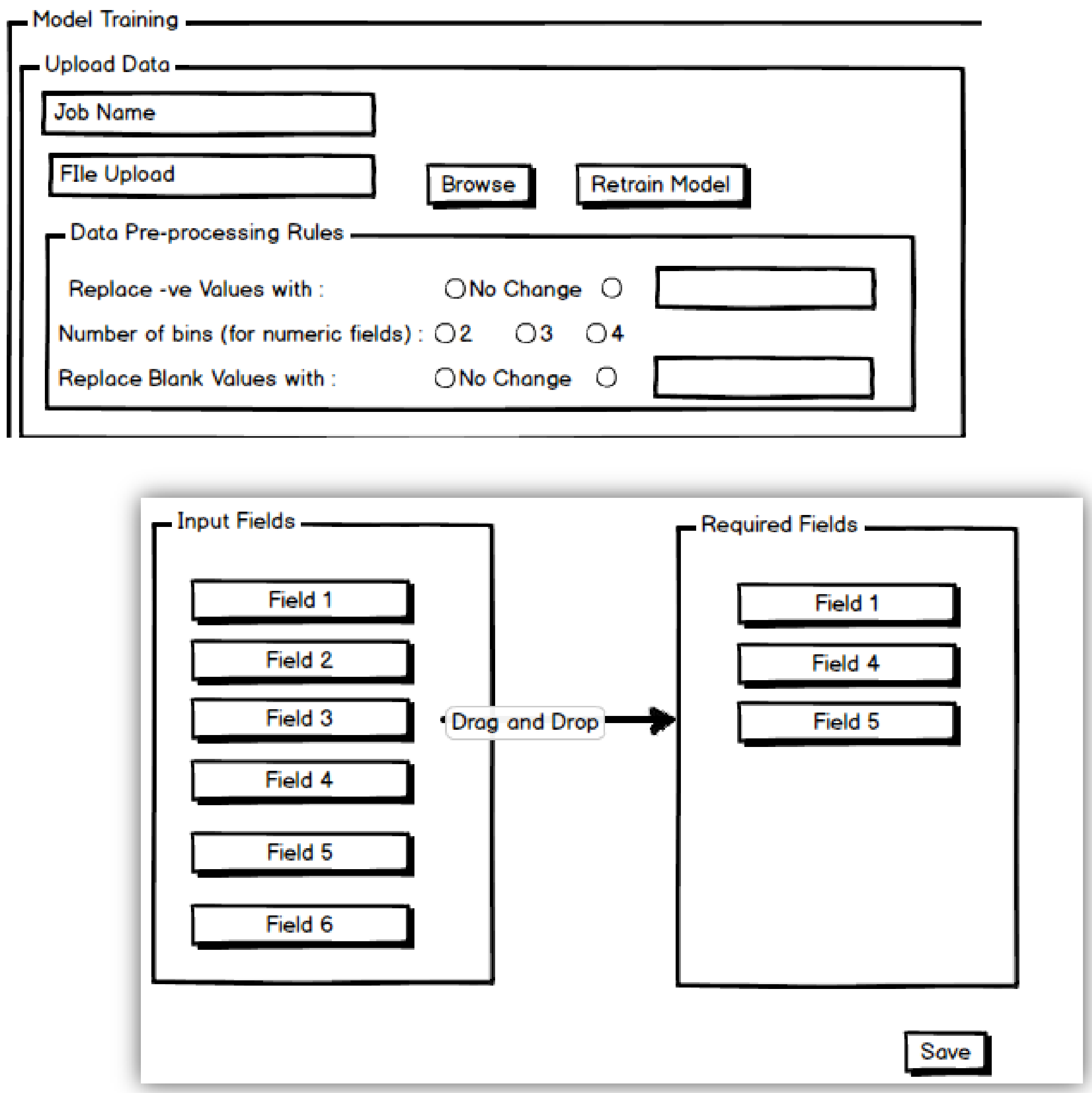


Figure 4: Pre-processing Rules and Field Selection Mockup

The first column of the grid shows the field (excel columns) that exist in the uploaded dataset. Users are able to select the Input field for the model training by dragging and dropping the fields from the left column to the right column.

3.1.2 Monitor Model Training Job Status

This section is (marked at Section 2 in the Main Page mockup) is used to show the status of data upload as well as the status of latest model training job that submitted by the user. It also shows the list of various models that were evaluated for this job, and the recommended model. Following figure shows the controls in Section #2 when the page initially loads.

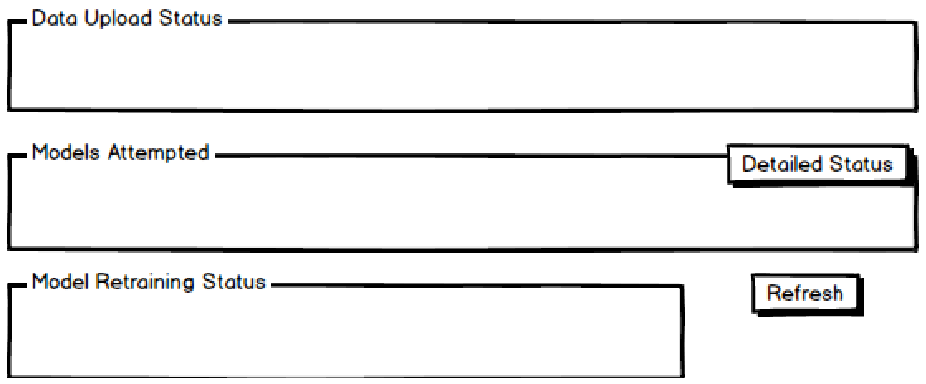
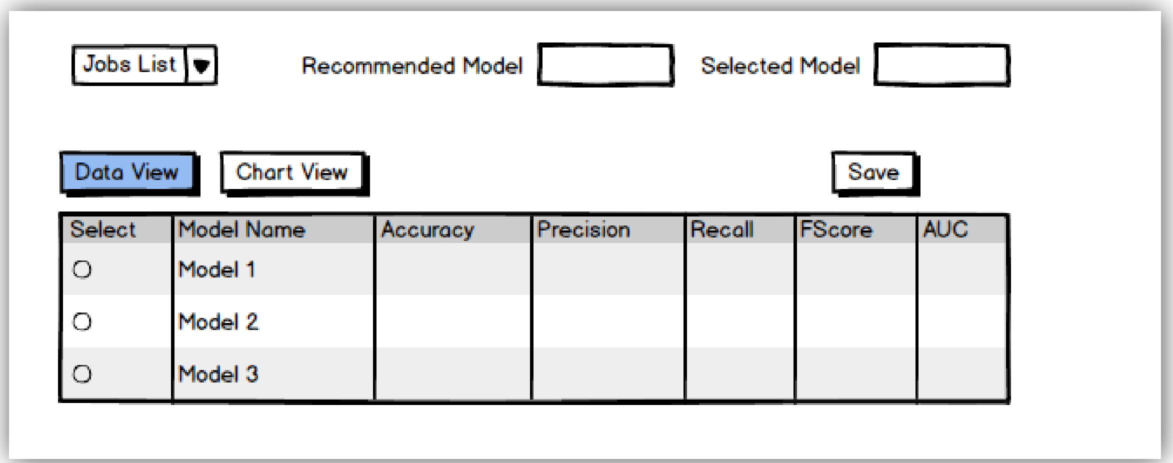


Figure 5: Section 2 - Model Training Status Mockup

The button titled “Detailed Status” should bring up a dialog box (mockup shown below) that shows statistical output related to the various models (such as Logistic Regression, Decision Tree, Support Vector Machine, etc) that were evaluated for this job.



Select	Model Name	Accuracy	Precision	Recall	FScore	AUC
○	Model 1					
○	Model 2					
○	Model 3					

Figure 6: Detailed Model Retraining Status – Data View Mockup

The dropdown titled "Jobs List" allows user to see the Statistical output related to model evaluation for the latest model training job, as well as any previous jobs that the user would have run. "Recommended Model" textbox shows the model recommended by the system, and "Selected Model" shows model that was selected by the user to override the recommended model. If user has not overridden the recommended model, "Selected Model" textbox will be shown same value as the "Recommended Model" textbox. A DataGrid shows the list of models that were trained for the uploaded data along with the statistical output that were used for evaluation (and comparison) of the models.

The grid view is the default view for this dialog box. The radio button in the first column titled "Select" shows the model recommended by the system, and can be used to select a different model to be used for model prediction. User clicks the "Save Changes" button to save the model selection.

Additionally, user can switch to the Chart View by clicking the "Chart View" button as shown below:

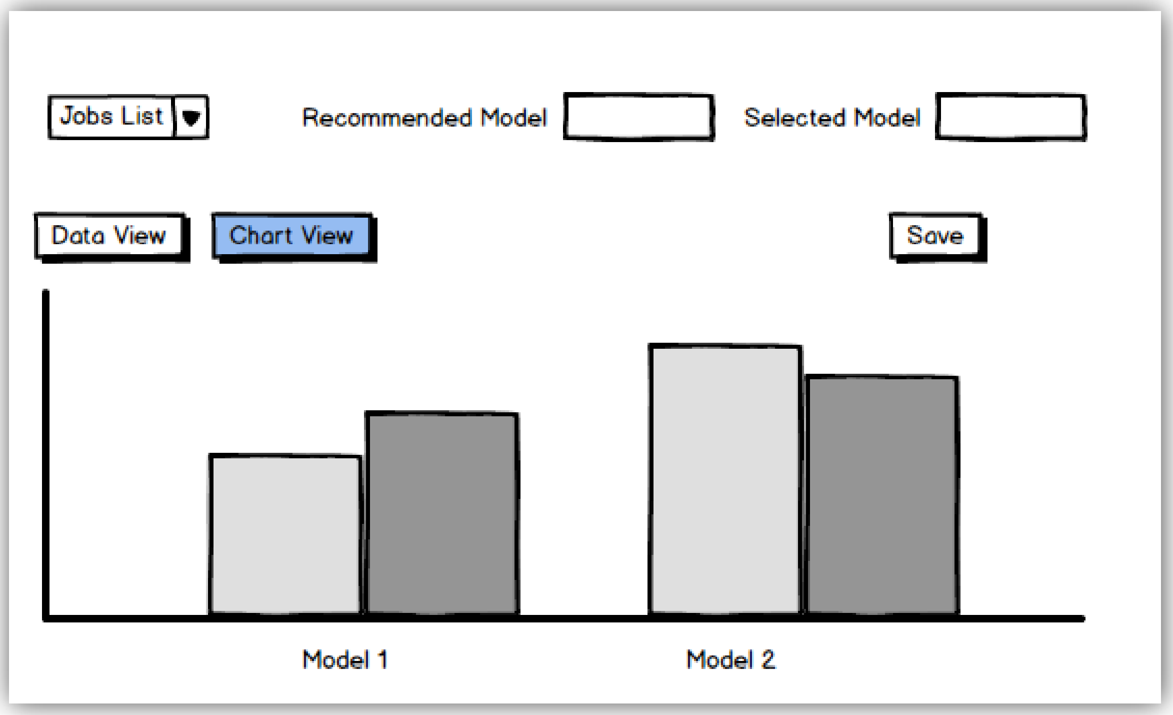


Figure 7: Model Training Status - Chart View Mockup

Chart View enables a user to compare various output related to model retraining in an intuitive fashion.

A short primer on the Statistical output data from model training jobs, including the model evaluation and comparison logic, is provided in the "Application Flow" section below.

3.1.3 Model Execution

This section (marked at Section 3 in the Main Page Mockup) is used to predict an outcome by providing input parameters. By default, it uses the latest trained model to do the prediction, but users have the option to select one of the previously trained models to do the prediction (and compare the outcome). The section looks like below (with no input parameters) when initially the page loads:

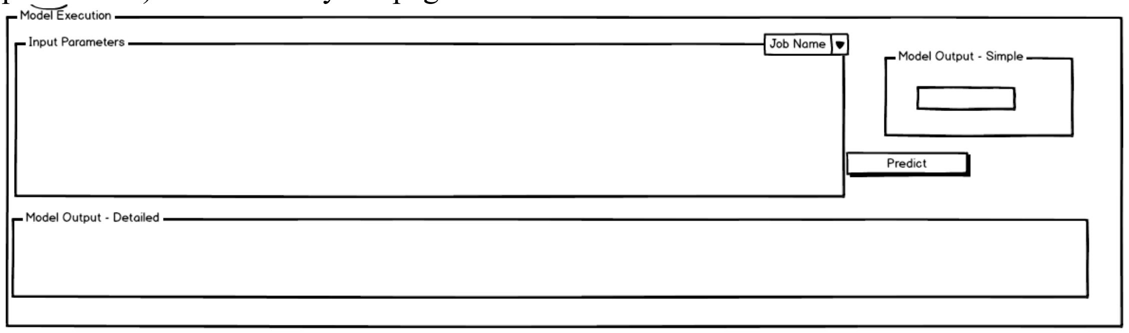
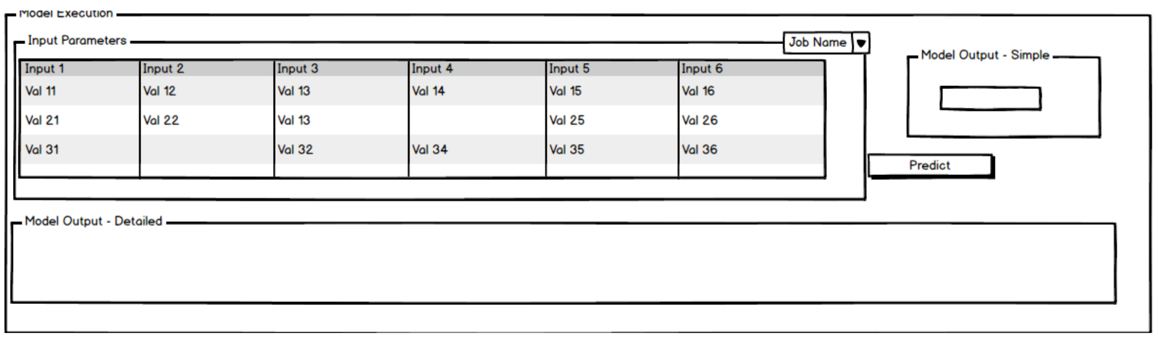


Figure 8: Model Execution - Section 3

The section "Input Parameters" is empty as no model training job has been created yet. Users can select one of the previous jobs from the dropdown, or create a new job (in section 1). On selecting a particular job from the dropdown, the "Input Parameters" controls is dynamically loaded with the datalist control, dependent upon the data uploaded by the user or the columns/fields selected by the user for model training (in section 1). Following image shows an example of Input Parameters section rendered based on the fields selected in the previous screenshot shown in Figure # 4 above.



Input 1	Input 2	Input 3	Input 4	Input 5	Input 6
Val 11	Val 12	Val 13	Val 14	Val 15	Val 16
Val 21	Val 22	Val 13		Val 25	Val 26
Val 31		Val 32	Val 34	Val 35	Val 36

Figure 9: Model Execution – Model Inputs Rendered Mockup

User can select the value from the datalist controls, and click the "Predict" button. At this point, system uses the "Selected Model" for the given job, and shows the prediction outcome in the "Model Output" textboxes, as shown in Figure 10.

"Model Output –Simple" textbox shows the output in a simple format (For example, 1 or 0 for a classification job) whereas "Model Output –Detailed" shows complete details of the input sent to the machine learning service, and the output received (including technical details such as "Scored Probabilities").

3.2 Application Flow

Following section presents the user flow for Predikt as well as Use Cases from two different domains.

The diagram below shows the typical user flow in Predikt:

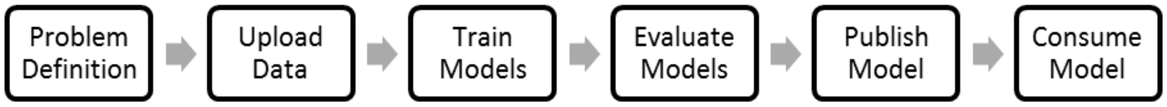


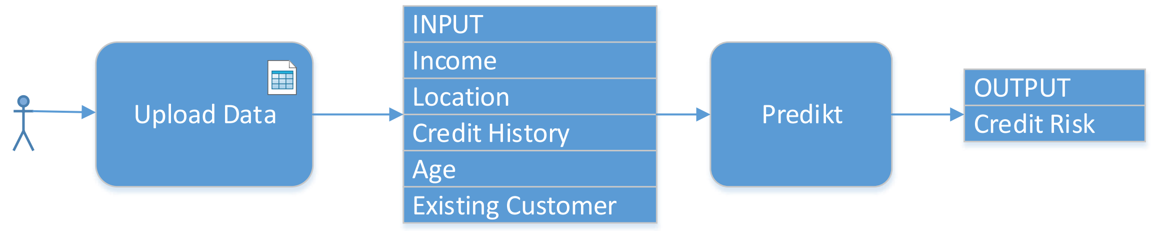
Figure 10: Application Flow

Following section dives deeper into each step:

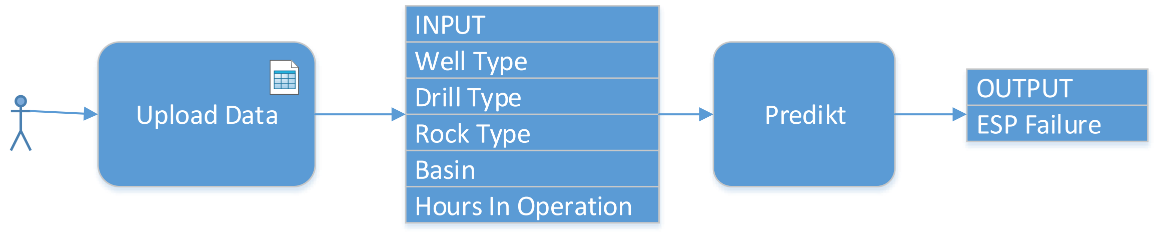
a. Problem Definition:

This is a one-time initial setup that will be done by customers to configure Predikt to meet their objectives and requirements. This step includes:

i. **Use Case Definition:** This is a domain specific step when user will need to decide the input variables (features/instances) as well as the output variable (outcome/classes) that they would want to predict. Following diagram presents use cases from two different domains.



Use Case 1: Financial (Credit Risk Evaluation)



Use Case 2: Oil & Gas (Electrical Submersible Pump Failure Prediction)

Figure 11: Use Cases

The first use case is for the financial domain, where a lending company is trying to evaluate the *Consumer Credit Risk* by training the model with input parameters such as *Income, Location, Credit History, Age* and *Existing Customer*.

The second use case is from Oil & Gas domain, where an E&P (Exploration and Production) company is trying to predict the probability of ESP (Electrical Submersible Pump) failure by training the model with input parameters such as *Well Type*, *Drill Type*, *Rock Type, Basin* and *Hours in Operation*.

ii. **Input data format setup:** Users will specify the format of data that will be provided to Predikt on a regular basis. This will include file format (excel, CSV, real-time database connection) as well as fields that they will provide in the file.

iii. Pre-processing Rules:

Users will be able to setup pre-processing rules such as number of bins for numeric columns, how –ve and null values will be handled in the application, etc).

Predikt provides an intuitive UI for users to perform this initial step for data input as well as pre-processing rules setup.

b. Upload Data:

This is the key responsibility for the customer on an ongoing basis. For Predikt outcome (prediction) to be accurate and relevant one of the expectation from the customer is that latest data will be provided on a frequent basis. This will help Predikt automation engine to get trained on the latest data, and outcome will reflect the latest trend in the customer's domain.

Predikt currently supports data upload using Excel and CSV files as well as exposes web services to support automated data upload.

c. Train Models

In this step, Predikt will apply various modeling techniques to the provided data and calibrate the model parameters to the optimal values. The output of this step is typically in the form of Statistical output such as Precision, Accuracy, Recall, FScore, etc (explained in the next section).

Predikt completely abstracts the end users from the technical complexity of training various models.

d. Score Models

Evaluation of various models based on output parameters, and be able to recommend the most optimal algorithm to maximize user's return is one of the core value proposition of Predikt. Users should be abstracted from the implementation differences between various models and underlying complexity. User's choices (if any) and level of interaction with the framework should be in context of domain. For example, the information given to users should be in the form of "Improvement in Prediction Outcome in one model Vs Other" as opposed to "chi-square test result of one model Vs other".

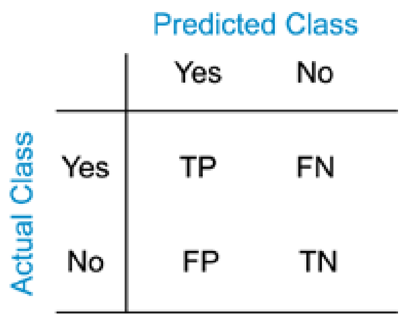
Following is a short introduction to some of the common evaluation metrics for a classification algorithm. Detailed discussion of various models and their respective metrics is out of scope of this report.

Model Performance Evaluation:

Although there can be many different metrics to measure the performance of a classifier, some of the most common ones try to estimate the Model performance based on correctness of prediction. They are:

1. True Positives (TP)
2. True negatives (TN)
3. False positives (FP)
4. False negatives (FN).

Following **Confusion Matrix** can help us understand these concepts much better:



Actual Class		Predicted Class	
		Yes	No
	Yes	TP	FN
	No	FP	TN

Figure 12: Confusion Matrix

Positives and Negatives reflect the two possible outcomes in a 2-class classifier algorithm. For example, for the financial lending use case explained above, the Positive outcome can be that the requester is credit-worthy whereas negative outcome can be that requester is not credit-worthy.

- True positives and True Negatives are the observations which were correctly predicted (out of total observations).
- A false positive is an error in which a predicted outcome improperly indicates presence of a condition, such as “credit-worthy” (the result is positive), when in reality it is not, while a false negative is an error in which a test result improperly indicates a negative outcome (not credit-worthy), when in reality it is present.

Although correctness of prediction measures explained above are basis for many other output, the model training output for end users is typically expressed in below terms:

- **Accuracy:** Accuracy is perhaps the most intuitive performance measure. It is simply the ratio of correctly predicted observations.

- **Precision:** Precision looks at the ratio of correct positive observations. The formula is:

  True Positives / (True Positives + False Positives).

- **Recall:** Recall is also known as sensitivity or true positive rate. It is the ratio of correctly predicted positive events. Recall is calculated as:

  True Positives / (True Positives + False Negatives).

  Note that the denominator is the count of all positive events, regardless whether they were correctly predicted by the model.

- **F1 Score:** The F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account, and is a good reflection of the model performance.

- **ROC Curve:** The curve is created by plotting the true positive rate against the false positive rate at various threshold settings. The threshold setting is the cut-off mark if the classifier output is not a choice between two labels, but is a real value (continuous output), to determine the outcome.

- **AUC:** AUC stands for "area under curve", and as it is name implies, it refers to the amount of area under the ROC curve, which theoretically is a value between 0 and 1. AUC metric is a good tool to compare multiple learning models. Since ROC curves of two models usually do not cross each other, hence when comparing two models, the one with a higher AUC will be the better one regardless of the threshold setting. Compared to the statistical measures of accuracy, precision, recall and F1 score, AUC is independence of threshold makes it uniquely qualified for model selection.

- **Lift:** Gain or lift is a measure of the effectiveness of a classification model calculated as the ratio between the results obtained with and without the model.

By default, Predikt uses AUC for model scoring and recommendation, but can be configured to use any of the other measures. It also allows for end users to override the recommended model after evaluating other performance measures.

e. Publish Model

Once various models have been evaluated for the best return, the next step in the process is to publish (deploy) the models to a production environment. This step makes the models available to end users for consumption.

Predikt completely abstracts the end users from the technical complexity of deploying and making the model available for consumption.

f. Consume Model

The final step in the process is the consumption of published model by end users.

Predikt facilitates this step by:

i. Providing UI for users to select input parameters and be able to see the prediction outcome.

ii. Providing web services that users can consume directly from their custom application.

3.3 Technical Design

Following section presents the high level architecture for Predikt and dives deeper into various components.

3.3.1 High Level Architecture

Predikt follows a 4-tier architecture as shown below:

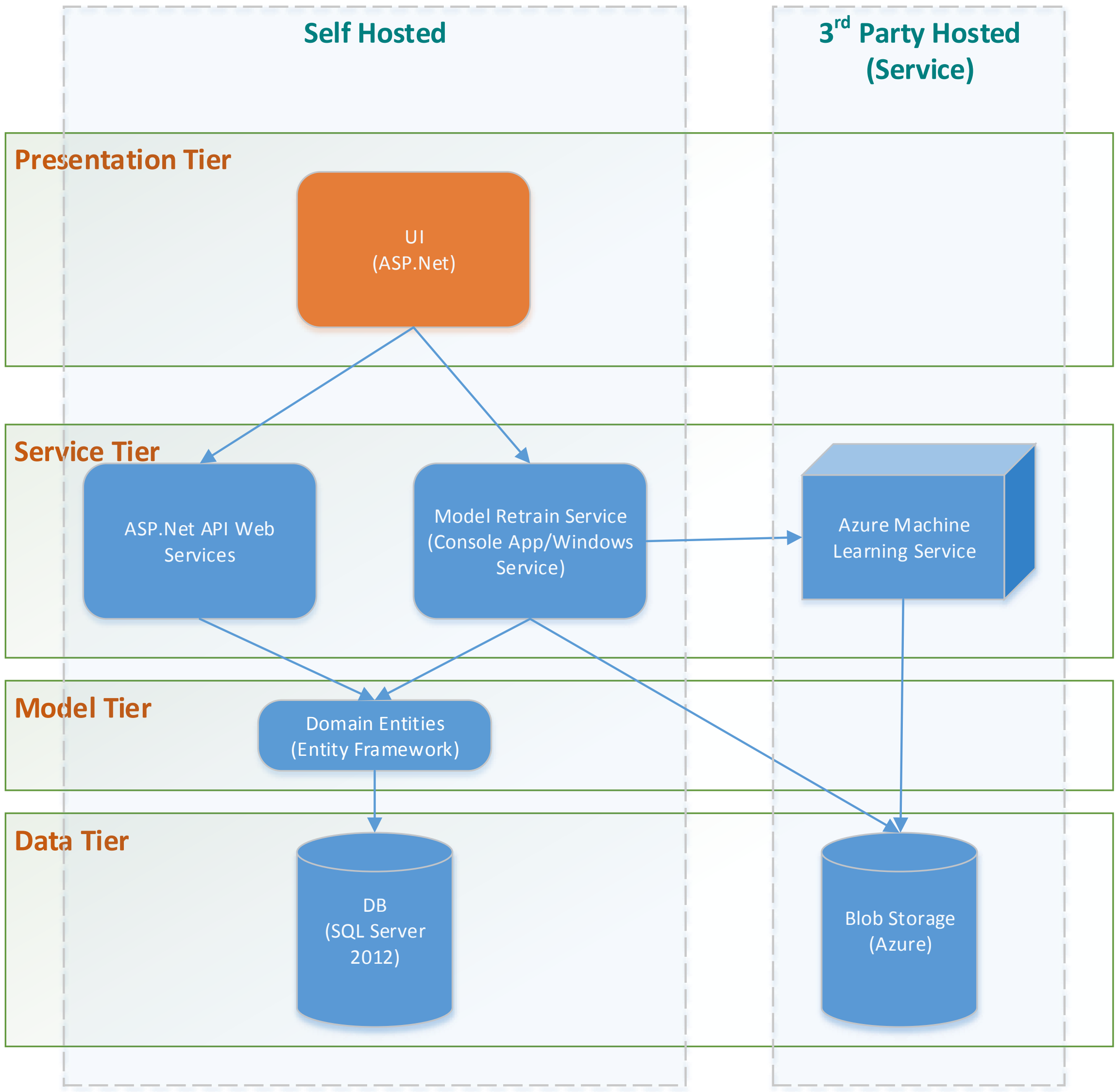

Self Hosted
3rd Party Hosted (Service)
Presentation Tier
UI (ASP.Net)
Service Tier
ASP.Net API Web Services
Model Retrain Service (Console App/Windows Service)
Azure Machine Learning Service
Model Tier
Domain Entities (Entity Framework)
Data Tier
DB (SQL Server 2012)
Blob Storage (Azure)

Figure 13: Technical Architecture

Following table provides a short overview of each of the tiers in the architecture:

Tier	Purpose	Technology
Presentation	User Interface Implementation	ASP.Net 4.5, Bootstrap 3.0
Services	Business Logic Implementation	ASP.Net Web Services API, Windows Console Application, Azure Machine Learning Service
Model	Object Relational Mapper for domain driven development	Entity Framework 6.0
Data	Storage	SQL Server 2012, Azure Storage

Table 1: Architecture Tier Definition

Design for each of the tiers is presented below:

3.3.2 Presentation Tier

Predikt provides an intuitive user experience built using ASP.Net 4.5. Bootstrap 3.0 is used to create a responsive UI. Following diagram presents the high level pattern for processing of ASP.net pages by .Net Framework:

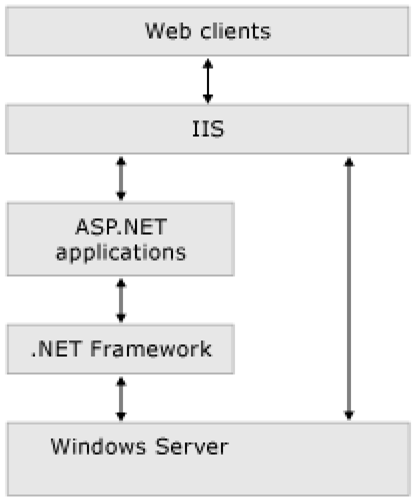


Figure 14: ASP.Net Application Processing

Steps shown in the diagram above are standard for any ASP.net application. Detailed description for each module is out of scope for this report.

3.3.3 Service Tier:

Service Tier comprises of the following three components:

i. ASP.Net Web Services API

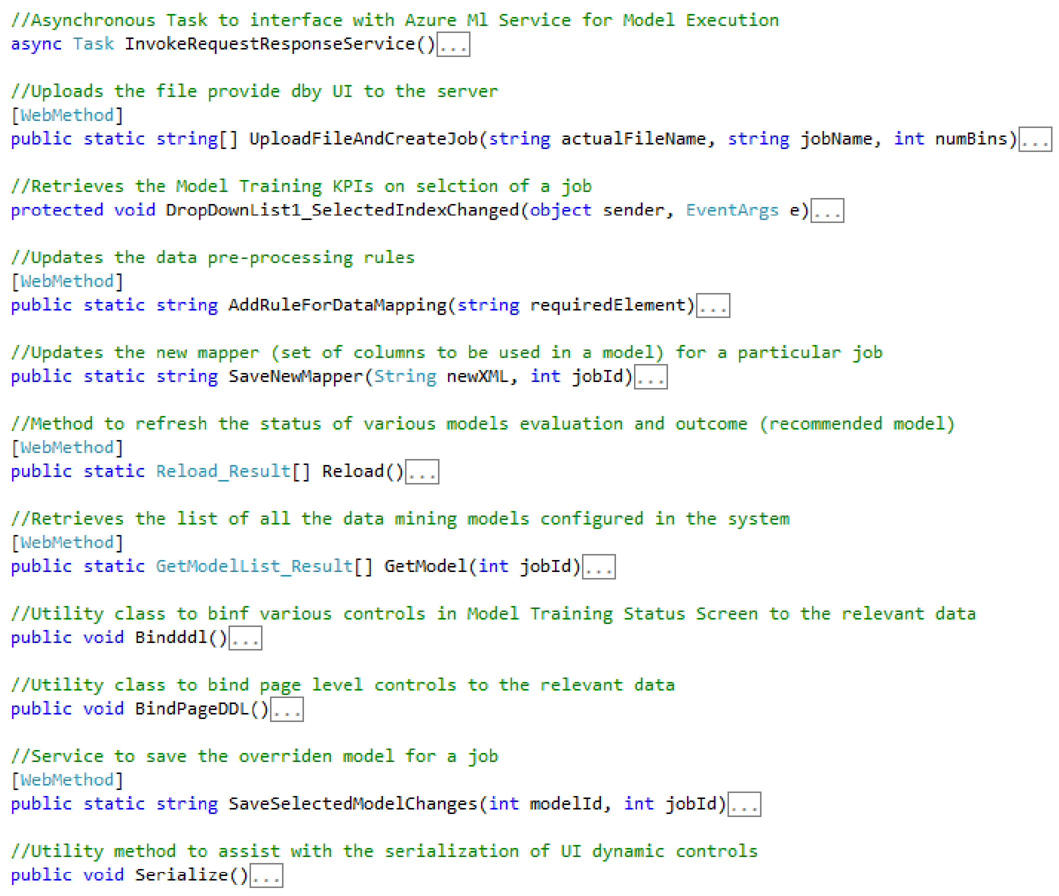
ii. Model Retrain Service

iii. Azure Machine Learning Service

i. **ASP.Net Web Services API** was used to achieve clear packaging of various operations & function which the UI requires to perform. The application is designed to keep the services layer loosely coupled and unaware of the consumption from the presentation tier. This strategy enabled us to open up the service layer to external systems by publishing web services for direct consumption by the client application (without going through our presentation layer).

This architecture provides for a uniform & standard way to deliver machine learning services to users through different presentation mediums like Mobile Apps in future.

Following figure shows some of the key web services exposed by this tier:

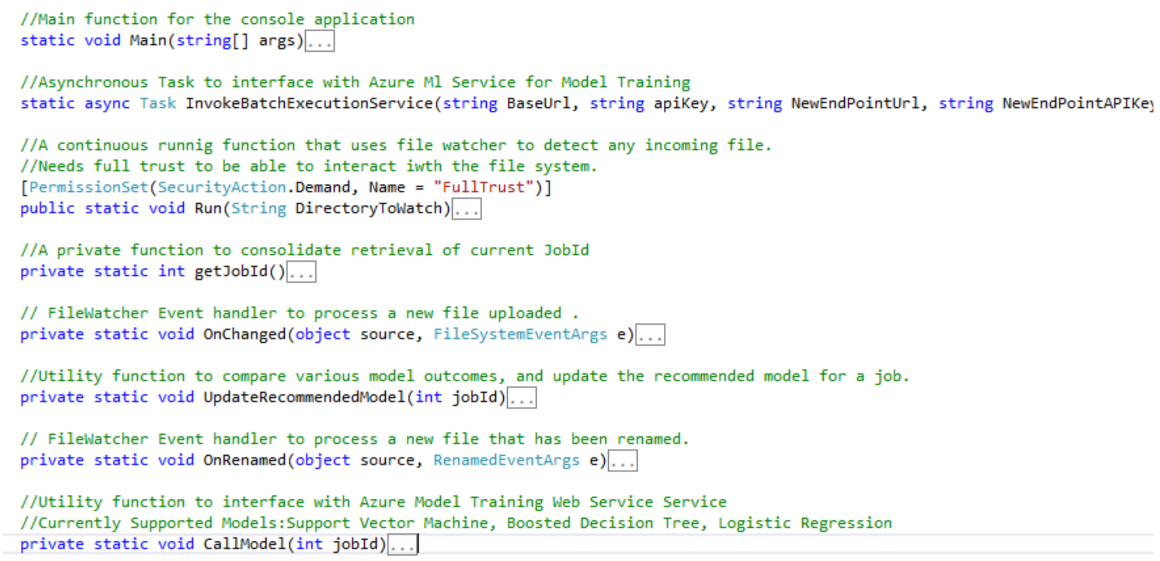


```
//Asynchronous Task to interface with Azure Ml Service for Model Execution
async Task InvokeRequestResponseService()...

//Uploads the file provide dby UI to the server
[WebMethod]
public static string[] UploadFileAndCreateJob(string actualFileName, string jobName, int numBins)...

//Retrieves the Model Training KPIs on selction of a job
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)...

//Updates the data pre-processing rules
[WebMethod]
public static string AddRuleForDataMapping(string requiredElement)...

//Updates the new mapper (set of columns to be used in a model) for a particular job
public static string SaveNewMapper(String newXML, int jobId)...

//Method to refresh the status of various models evaluation and outcome (recommended model)
[WebMethod]
public static Reload_Result[] Reload()...

//Retrieves the list of all the data mining models configured in the system
[WebMethod]
public static GetModelList_Result[] GetModel(int jobId)...

//Utility class to binf various controls in Model Training Status Screen to the relevant data
public void Bindddl()...

//Utility class to bind page level controls to the relevant data
public void BindPageDDL()...

//Service to save the overriden model for a job
[WebMethod]
public static string SaveSelectedModelChanges(int modelId, int jobId)...

//Utility method to assist with the serialization of UI dynamic controls
public void Serialize()...
```

Figure 15: Key Methods in Asp.Net API Web Services Project

ii. **Model Retrain Service** is a Console Client that can also be deployed as a Windows Service. It takes the file that was uploaded by the end user with the latest data, and uploads it to Windows Azure Blob Storage. It also invokes the Azure ML Studio REST Web Service to start the model retraining. It also keeps track of the status of model retraining, and provides the intermediate messages along the way. On completion of evaluation, this service is also responsible for selection of the recommendation model for a particular job.

Following figure shows some of the key methods implemented in the Model retrain Service:



```
//Main function for the console application
static void Main(string[] args)...

//Asynchronous Task to interface with Azure Ml Service for Model Training
static async Task InvokeBatchExecutionService(string BaseUrl, string apiKey, string NewEndPointUrl, string NewEndPointAPIKey

//A continuous runnig function that uses file watcher to detect any incoming file.
//Needs full trust to be able to interact iwth the file system.
[PermissionSet(SecurityAction.Demand, Name = "FullTrust")]
public static void Run(String DirectoryToWatch)...

//A private function to consolidate retrieval of current JobId
private static int getJobId()...

// FileWatcher Event handler to process a new file uploaded .
private static void OnChanged(object source, FileSystemEventArgs e)...

//Utility function to compare various model outcomes, and update the recommended model for a job.
private static void UpdateRecommendedModel(int jobId)...

// FileWatcher Event handler to process a new file that has been renamed.
private static void OnRenamed(object source, RenamedEventArgs e)...

//Utility function to interface with Azure Model Training Web Service Service
//Currently Supported Models:Support Vector Machine, Boosted Decision Tree, Logistic Regression
private static void CallModel(int jobId)...
```

Figure 16: Key Methods in Model Retrain Service

iii. **Azure Machine Learning Service** is used as the backend Engine to configure and deploy various Machine Learning Algorithms. It is a SaaS (Software as a

Service) service provided by Microsoft that enables deployment of various Machine Learning Algorithms.

I decided to use Azure ML Studio for Predikt because it is a hosted service (and thereby eliminating the need to setup a separate infrastructure) and exposes web services that can be called from client applications to perform various common tasks such as:

i. Execute a deployed model
ii. Retrain the model

Azure ML Studio also supports Synchronous as well as Asynchronous options to call web services. This capability coupled with tight integration with Azure Storage Service can enable elastic cloud based scaling for high volume data mining requirements.

Following figure shows the orchestration for hosting a Two Class Boosted Decision Tree algorithm in Azure ML Studio Service:

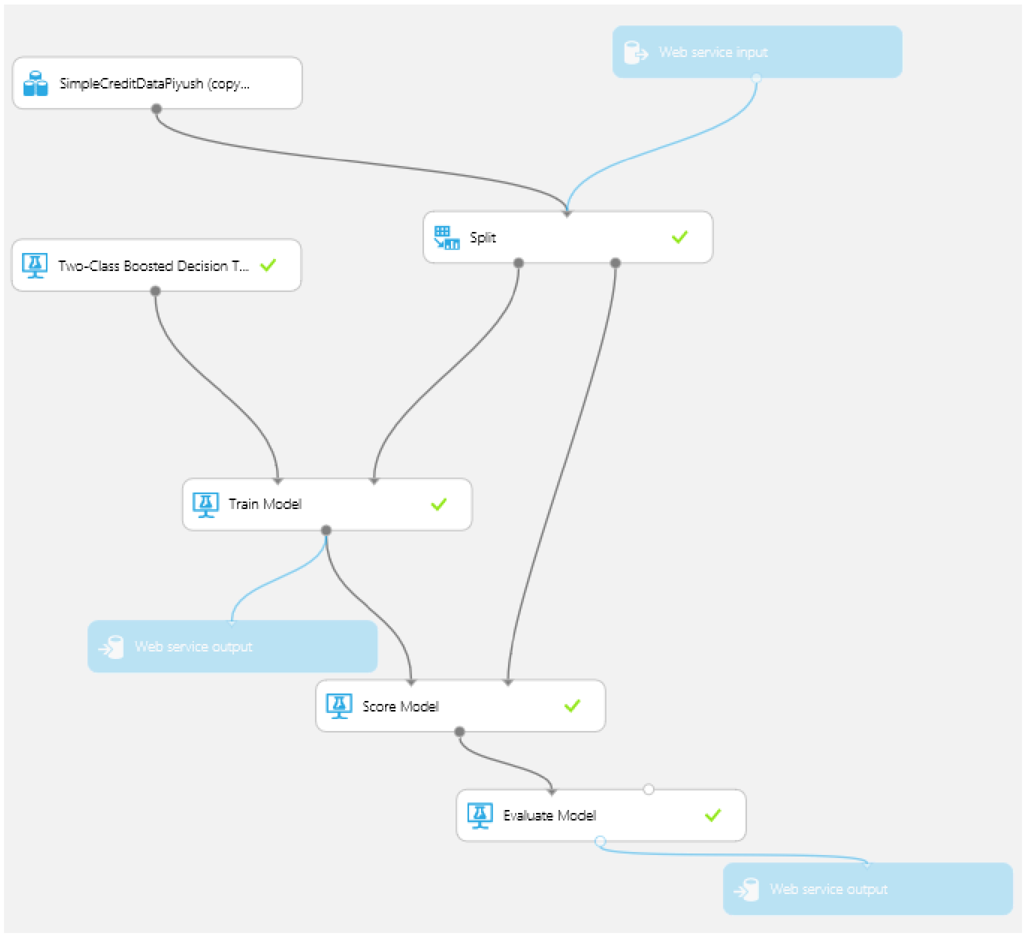


Figure 17: Azure ML Studio - Double Boosted Decision Tree Model

Although Azure ML Studio was used for this exercise, Predikt architecture has been designed to work with other open-source alternatives such as RApache, Shiny, OpenCPU, etc or commercial offerings such as SAS, IBM Watson Analytics, etc that can expose access to machine learning algorithms as web services.

3.3.4 Model Tier:

Predikt follows the principles of Model Driven Development (MDD) by enabling the separation of the Physical Model from the conceptual model which is exposed to higher layers for consumption. This limits the impact of changes in the physical model implementation and storage choices to the upper layers. The layers above are decoupled from the storage layer implementation details which leads to overall increase in the reliability and maintainability of the application. Apart from that it provides the data access consumer a more intuitive and easy to use domain entities based conceptual model instead of a technical table entities based physical model.

Following figure presents the Entity Model for Predikt:

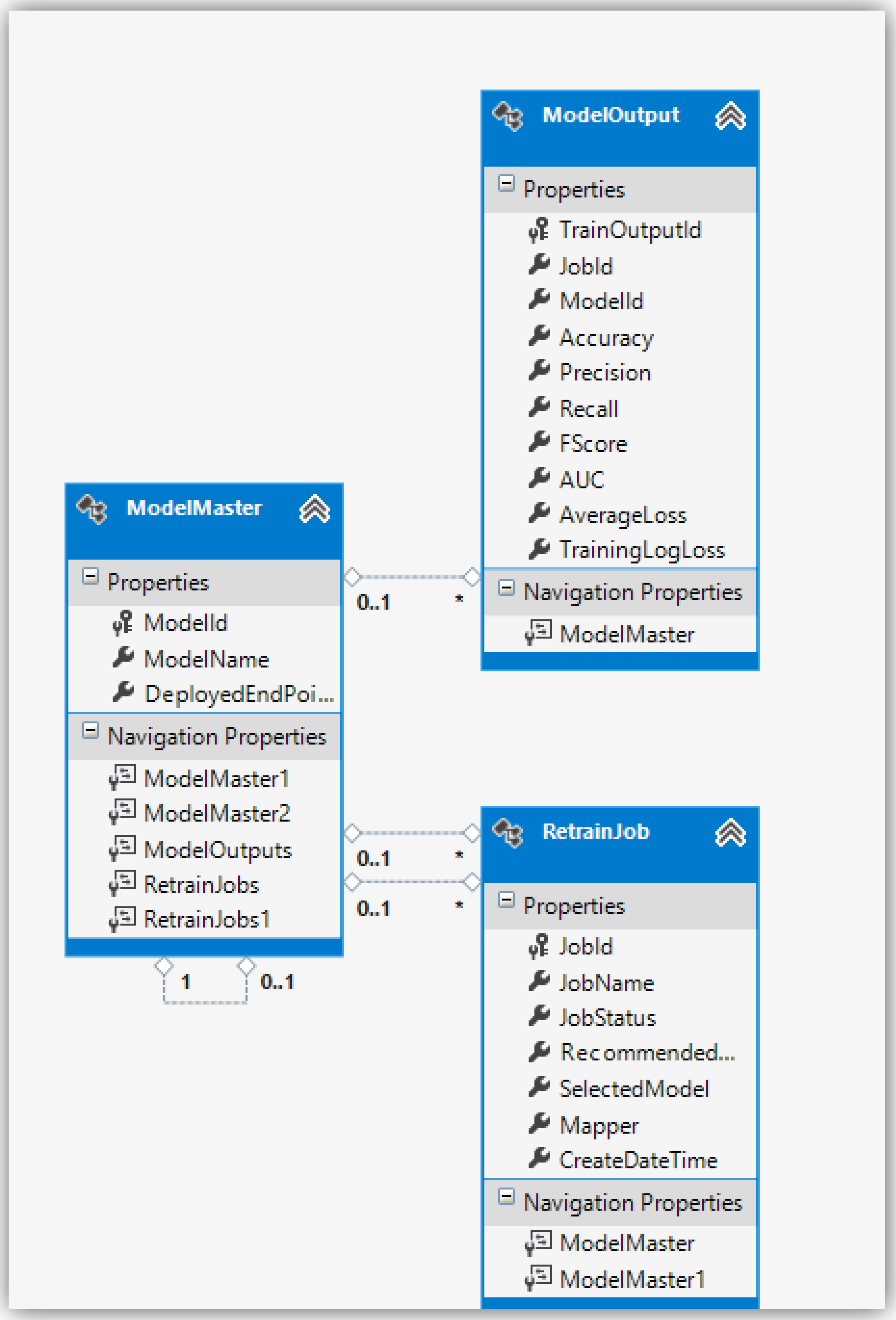


Figure 18: Predikt Entity Model

Microsoft Entity Framework 6.0 is used to implement the Model tier. Following diagram shows the conceptual architecture for the implementation of Entity Framework:

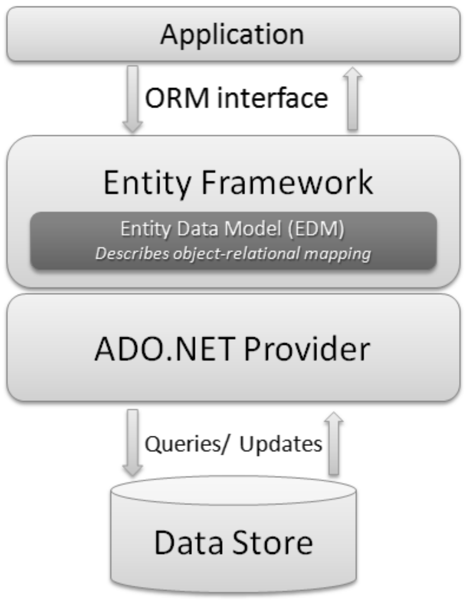


Figure 19: Entity Framework Conceptual Architecture

I decided to use an object-relational (ORM) mapper for Predikt (instead of direct database access) so that the development can be done with the dependency on domain-specific objects, instead of relational database structures. It allowed for rapid development of Services layer without me needing to update database layer for every change in UI and services layer. It also eliminated the need for me to write data-access code, and will allow in future to easily switch my database from SQL Server 2012 to something else (if needed).

3.3.5 Database Tier

Database Tier comprises of the following are the two components:

i. **SQL Server 2012:**

SQL Server 2012 is used as the relational database.

Following diagram shows the database model for Predikt.

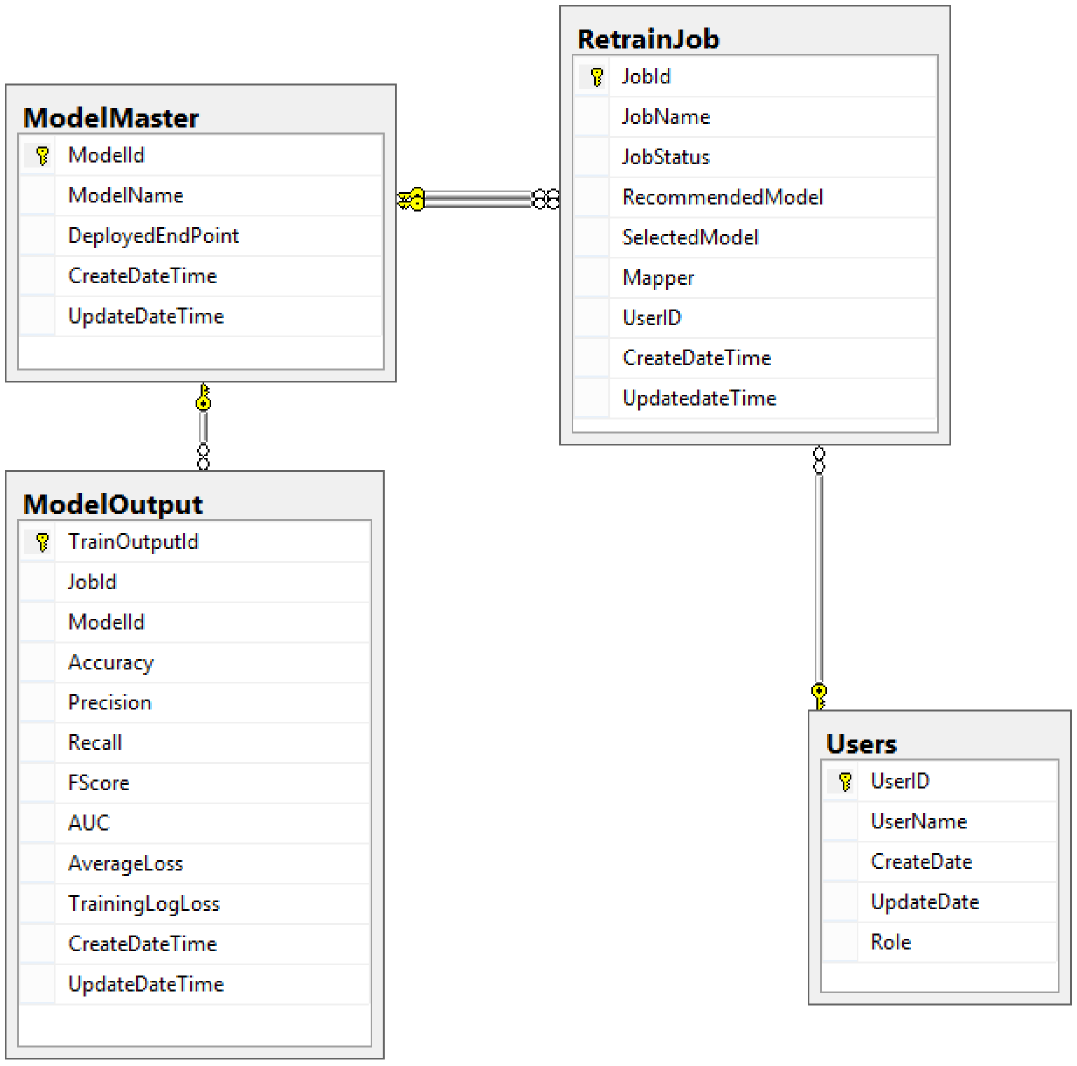


Figure 20: Database Design

ii. Blob Storage:

Windows Azure Storage was used for storing data such as:

i. Newly uploaded data

ii. Model Retraining Output

Microsoft Azure Storage is a scalable elastic cloud storage solution from Microsoft. Windows Azure Storage was selected for this sample application because of streamlined set of web services it exposes as well as because of tight integration with Azure ML Studio.

Azure Blob storage is one of the service in Azure Storage for storing large amounts of unstructured data, such as text or binary data, and can be accessed from anywhere in the world via HTTP or HTTPS.

Following figure contains shows the configuration of Blob Storage for Predikt:

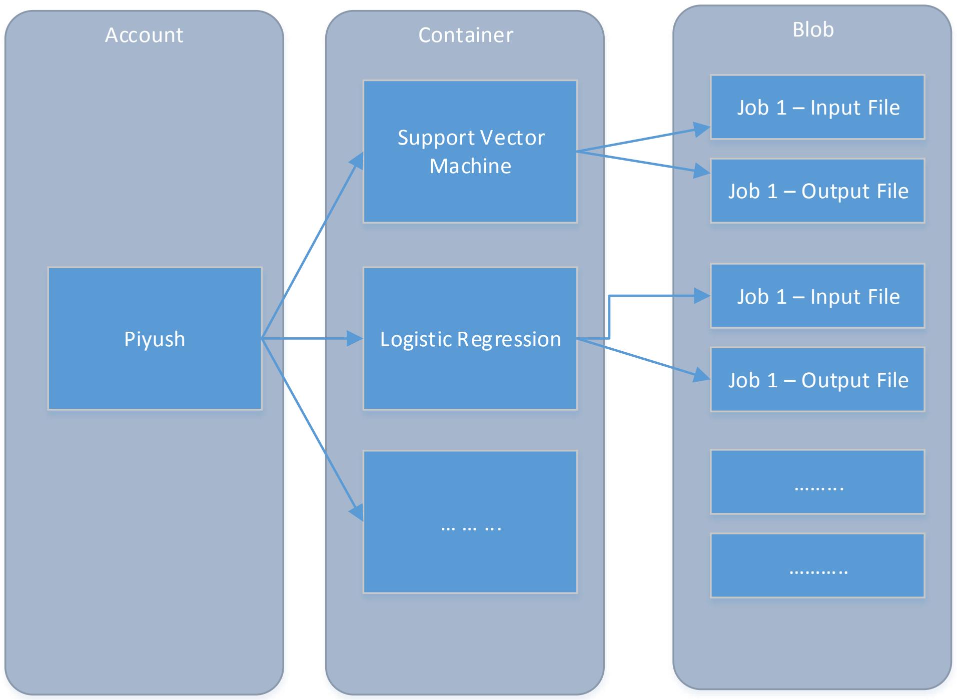


Figure 21: Predikt Blob Storage Setup

There are three main components in the diagram:

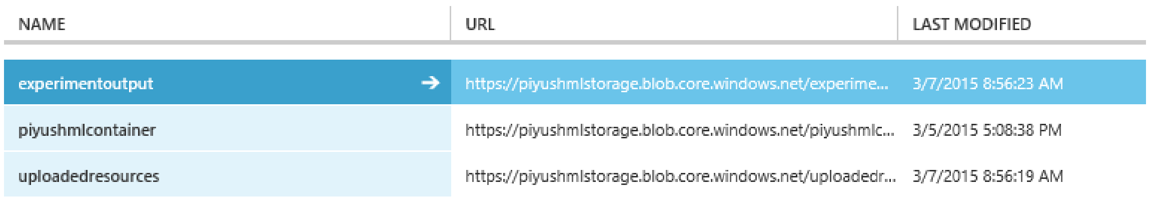
- **Storage Account:** All access to Azure Storage is done through a storage account. There is typically one account maintained for a product.
- **Container:** A container provides a grouping of a set of blobs. All blobs must be in a container. An account can contain an unlimited number of containers. A container can store an unlimited number of blobs.

  I decided to create a container per supported model (currently three) to allow Predikt to scale as the number of models grow. It will also allow Predikt to control the scalability on a per model basis. For example, if there are more takers for *Support Vector Machine* compared to *Logistic regression*, having separate container will allow up to scale up (buy better subscription) the storage for SVM in terms of capacity, availability and performance.

- **Blob:** Blob can be a file of any type and size. There are two types of blobs that can be stored in Azure Storage: block and page blobs. Most files are block blobs. A single block blob can be up to 200 GB in size. This tutorial uses block blobs. Page blobs can be up to 1 TB in size. Predikt is currently using Block Blob to store Input and Output files.

URL format: Blobs are addressable using the following URL format: h*ttp://<storageaccount>.blob.core.windows.net/<container>/<blob>*

Following is a screenshot of the three blob containers that were created for Predikt.



NAME	URL	LAST MODIFIED
experimentoutput →	https://piyushmlstorage.blob.core.windows.net/experime...	3/7/2015 8:56:23 AM
piyushmlcontainer	https://piyushmlstorage.blob.core.windows.net/piyushmlc...	3/5/2015 5:08:38 PM
uploadedresources	https://piyushmlstorage.blob.core.windows.net/uploadedr...	3/7/2015 8:56:19 AM

Figure 22: Azure Blob Storage URLs

Chapter 4: Project Results

Following chapter presents various qualitative as well as quantitative results from the project including screenshot and Software Engineering Metrics.

4.1 Screenshots

Following screenshot shows the main page when Predikt loads:

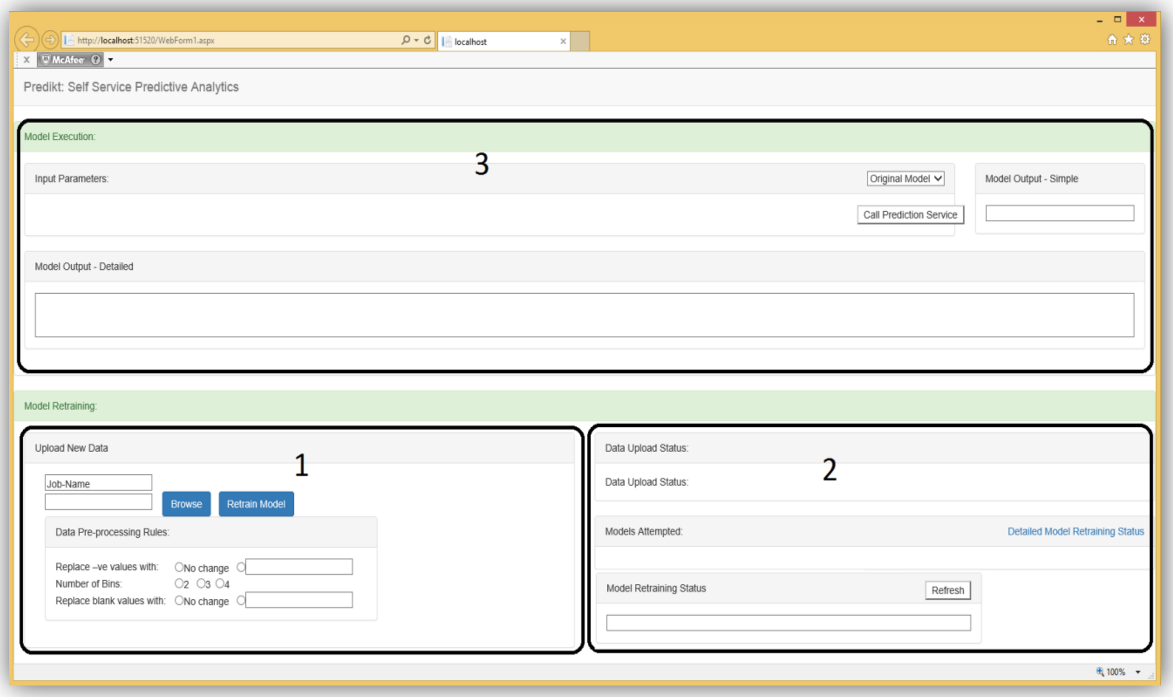
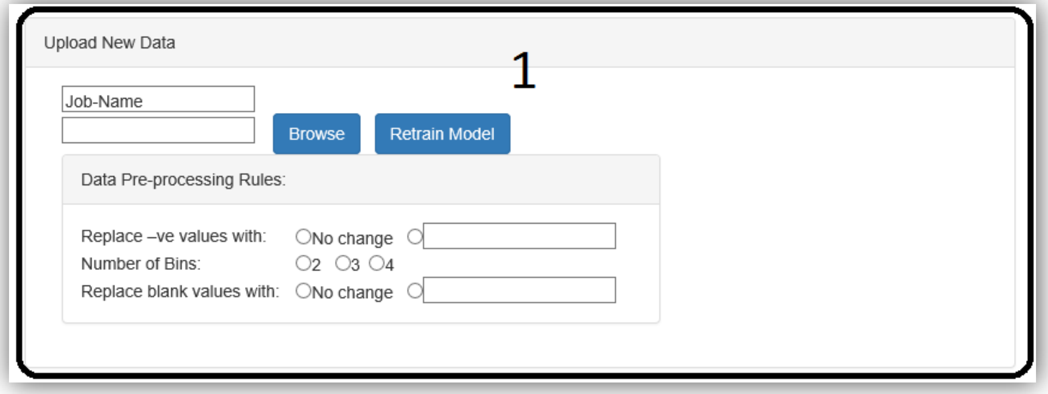


Figure 23: Predict Main Page

As envisioned during the design phase, there are three main sections in the screen:

i. Data Upload and Pre-processing rules Setup

This section is used by users to upload new data, and set data pre-processing rules as shown below:


Upload New Data

1

Job-Name

Browse Retrain Model

Data Pre-processing Rules:

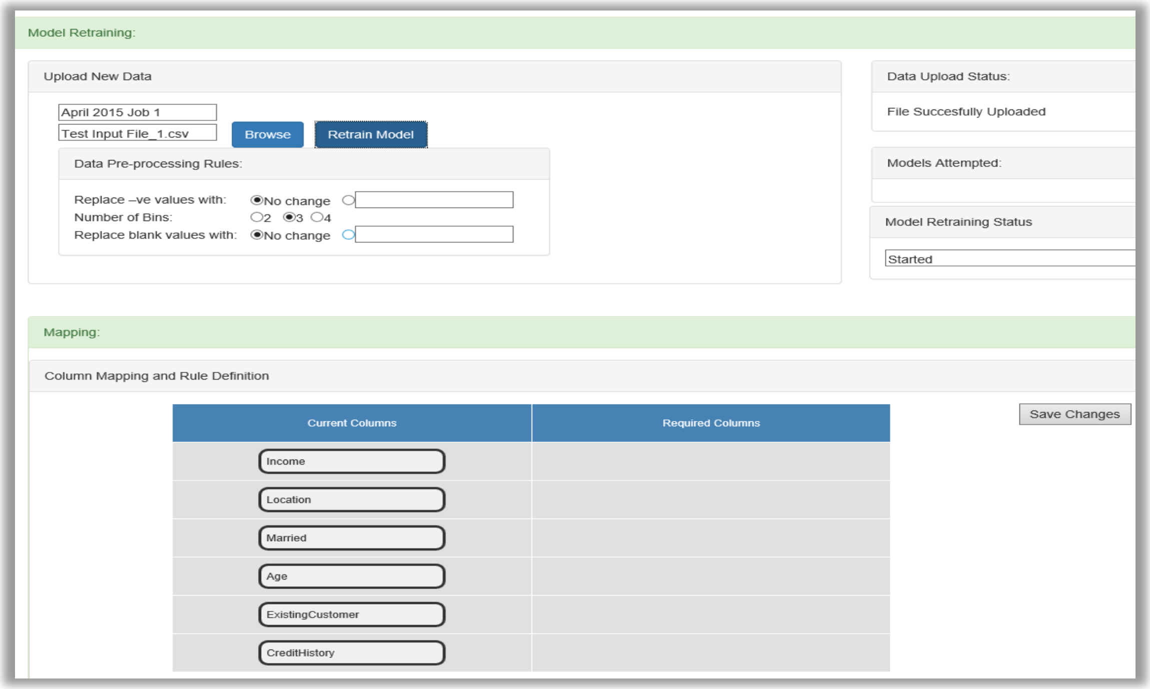
Replace –ve values with: ○No change ○

Number of Bins: ○2 ○3 ○4

Replace blank values with: ○No change ○

Figure 24: Data Upload (Section 1)

Following control is shown to select the fields from the new dataset that they would like to incorporate into the model (as Input Parameters). In this screenshot, user is trying to create a training job named *April 2015 Job 1* and is uploading a file named *Test Input File_1.CSV*.


Model Retraining:

Upload New Data

April 2015 Job 1

Test Input File_1.csv

Browse Retrain Model

Data Pre-processing Rules:

Replace –ve values with: [✓ No change] ○

Number of Bins: ○2 [✓ 3] ○4

Replace blank values with: [✓ No change] ○

Data Upload Status:

File Succesfully Uploaded

Models Attempted:

Model Retraining Status

Started

Mapping:

Column Mapping and Rule Definition

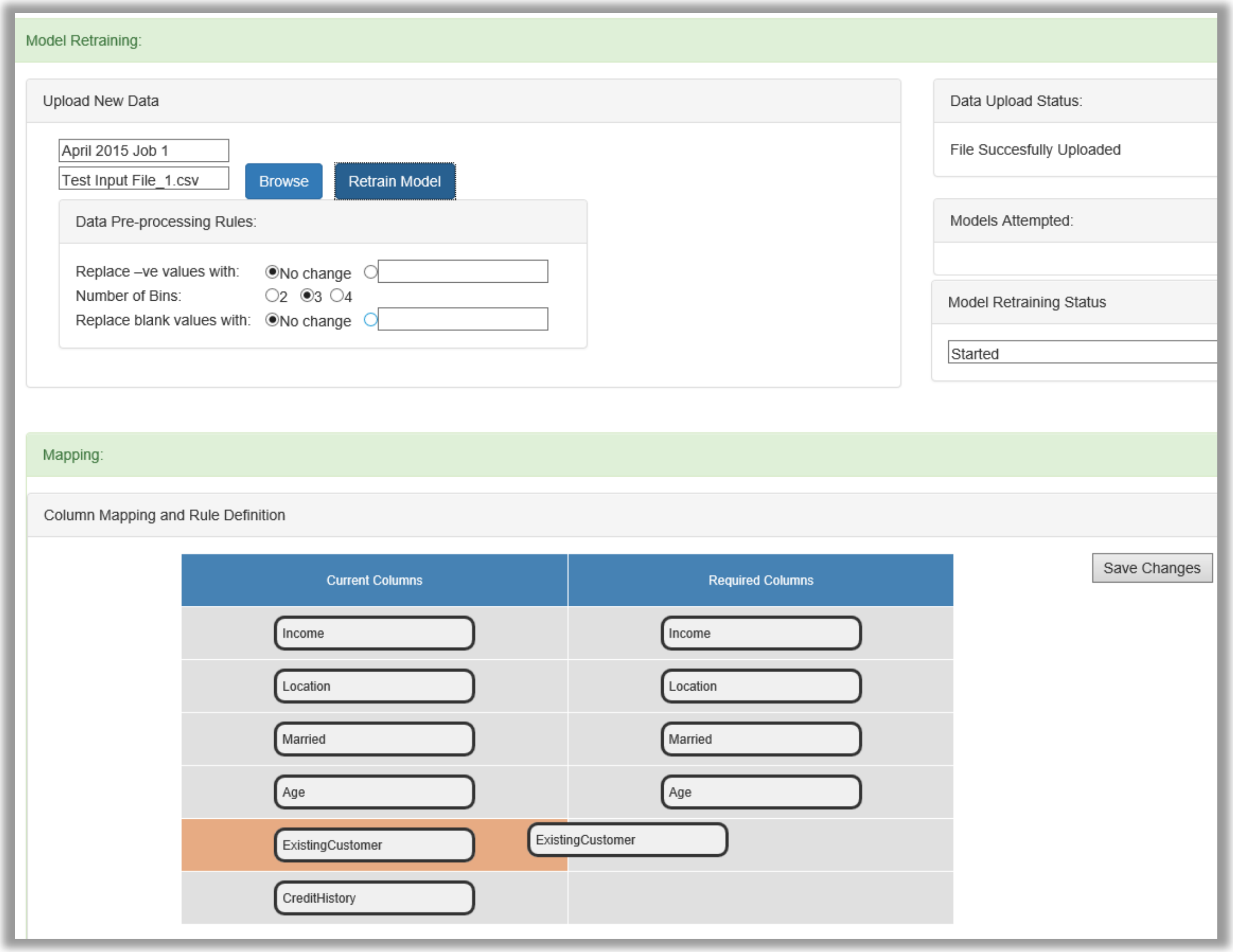
Save Changes

Current Columns	Required Columns
Income	
Location	
Married	
Age	
ExistingCustomer	
CreditHistory	

Figure 25: Pre-processing Rules and Field Selection

The first column of the grid shows the field (excel columns) that exist in the uploaded data – Income, Location, Married, Age, Existing Customer and Credit History.
As per the UI design, users are able to select the Input field for the model training by dragging and dropping the fields from the left column to the right column as shown below.

You will also notice that the *Data Upload Status* section on top right corner has been uploaded with the status of file upload, and *Model Retraining Status* has been updated to reflect the latest status.


Model Retraining:

Upload New Data

April 2015 Job 1

Test Input File_1.csv

Data Pre-processing Rules:

Replace –ve values with: [✓ No change] ○

Number of Bins: ○2 [✓ 3] ○4

Replace blank values with: [✓ No change] ○

Data Upload Status:

File Succesfully Uploaded

Models Attempted:

Model Retraining Status

Started

Mapping:

Column Mapping and Rule Definition

Current Columns	Required Columns
Income	Income
Location	Location
Married	Married
Age	Age
ExistingCustomer	ExistingCustomer
CreditHistory	

Figure 26: Field Selection

ii. Monitor Model Training Job Status

As per design, Section 2 is used to show the status of data upload as well as the status of latest model training job that submitted by the user. It also shows the list of various models that were evaluated for this job, and the recommended model. Following figure shows the controls in Section #2 when the page is initially loaded.

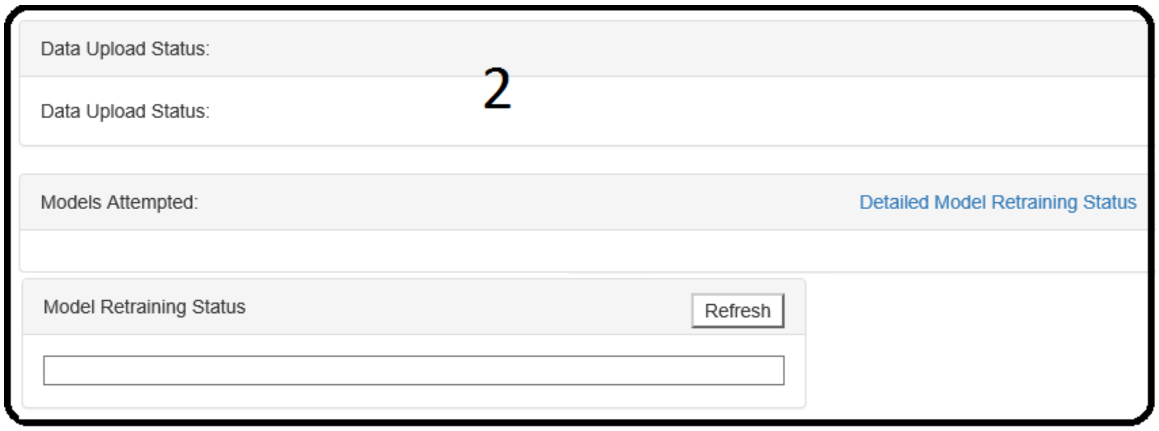


Figure 27: Model Training Status – Not Started (Section 2)

Following figure shows the controls in Section # 2 after a job has been run.

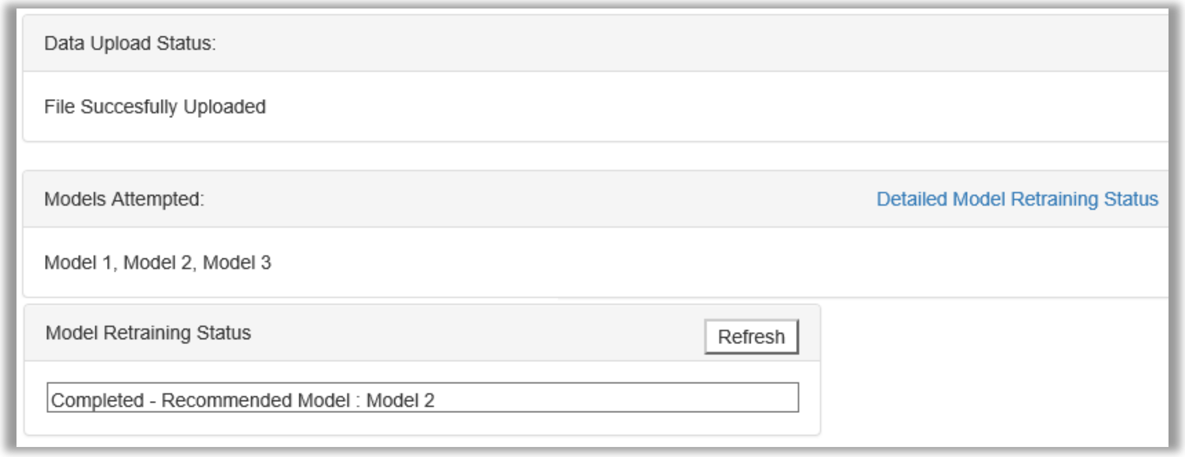
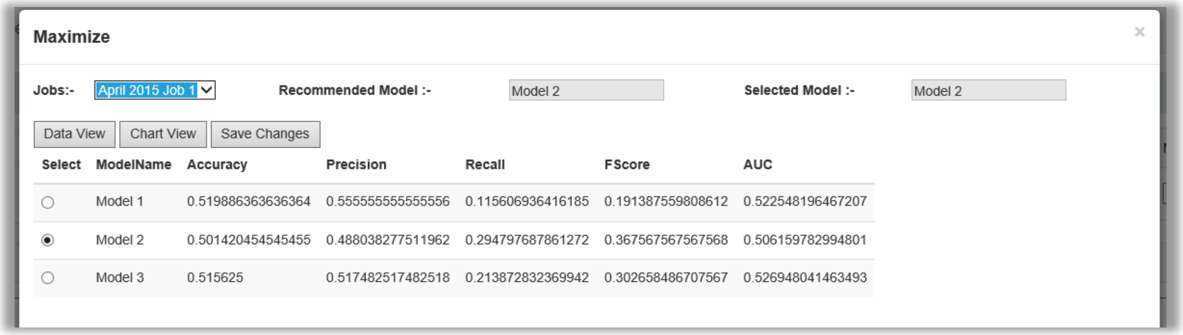

Data Upload Status:
File Succesfully Uploaded
Models Attempted:
Detailed Model Retraining Status
Model 1, Model 2, Model 3
Model Retraining Status
Refresh
Completed - Recommended Model : Model 2

Figure 28: Model Training Status – On Completion (Section 2)

The hyperlink titled "Detailed Model Retraining Status" brings up a dialog box (shown below) that shows statistical output related to the various models (such as Logistic Regression, Decision Tree, Support Vector Machine, etc) that were evaluated for this job.



Maximize

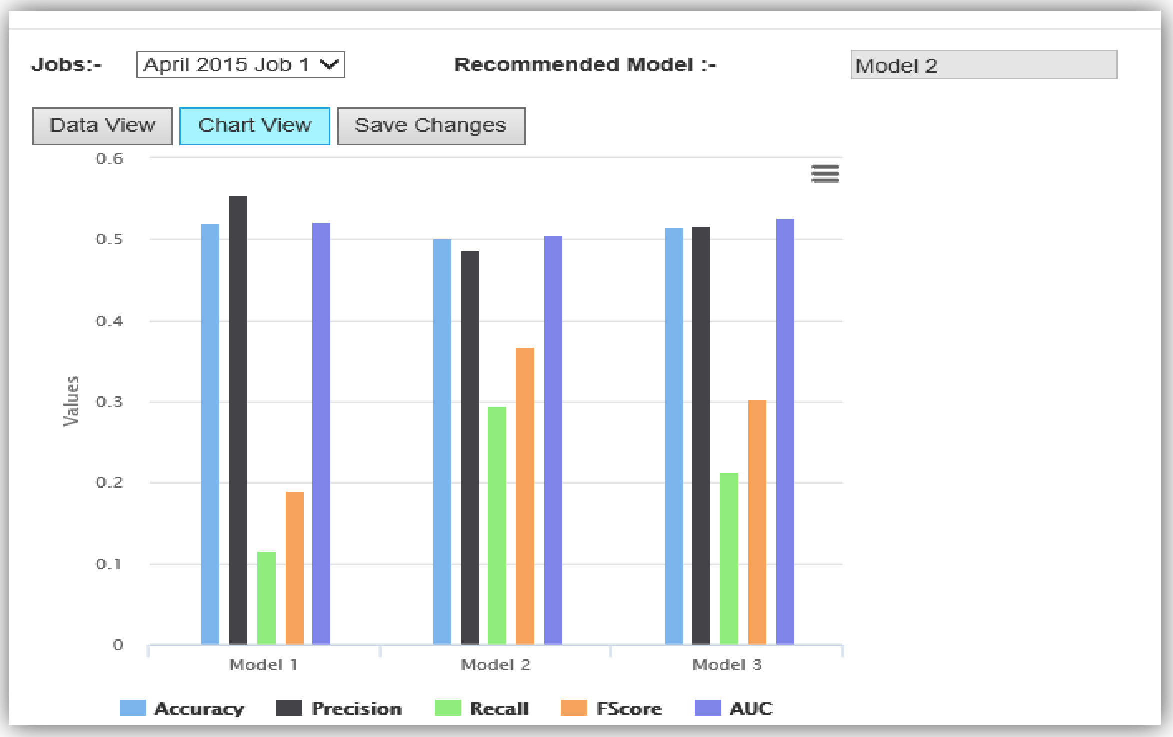
Jobs:- April 2015 Job 1 Recommended Model :- Model 2 Selected Model :- Model 2

Select	ModelName	Accuracy	Precision	Recall	FScore	AUC
○	Model 1	0.519886363636364	0.555555555555556	0.115606936416185	0.191387559808612	0.522548196467207
●	Model 2	0.501420454545455	0.488038277511962	0.294797687861272	0.367567567567568	0.506159782994801
○	Model 3	0.515625	0.517482517482518	0.213872832369942	0.302658486707567	0.526948041463493

Figure 29: Detailed Model Retraining Status

You will notice that the dropdown titled "Jobs" is selected by default as it was the most recent job created by the user. This screen is also showing the Statistical Output for the job *April 2015 Job 1*. Since user did not override the recommended model for this job, the *Recommended* as well as *Selected Model* fields are showing the same Model.

Following Chart View is rendered on click of the *Chart View* button:



Jobs:- April 2015 Job 1 Recommended Model :- Model 2

Figure 30: Model Training Status - Chart View

iii. Model Execution

As per design, Model Execution Section shown below can be used by users to predict an outcome by providing input parameters. The section looks like below when initially the page loads:

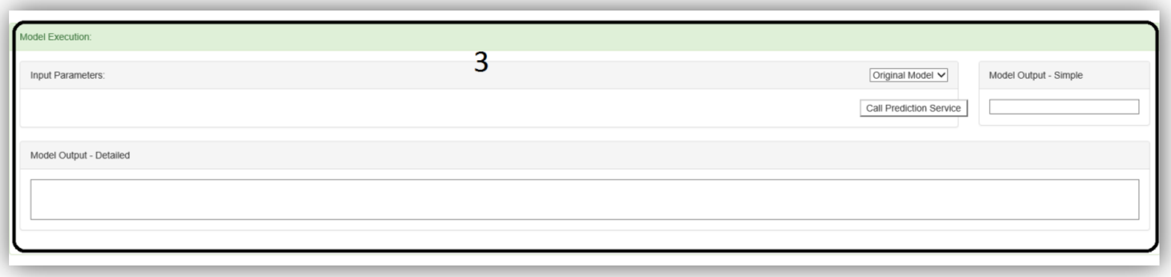


Figure 31: Model Execution Screenshot (Section 3)

Following screenshot shows an example of Input Parameters section rendered based on the fields selected in the previous screenshot:

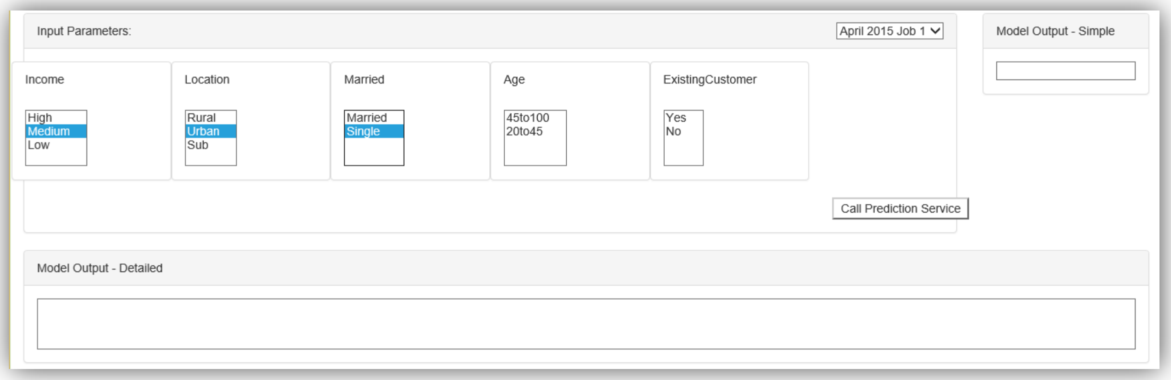
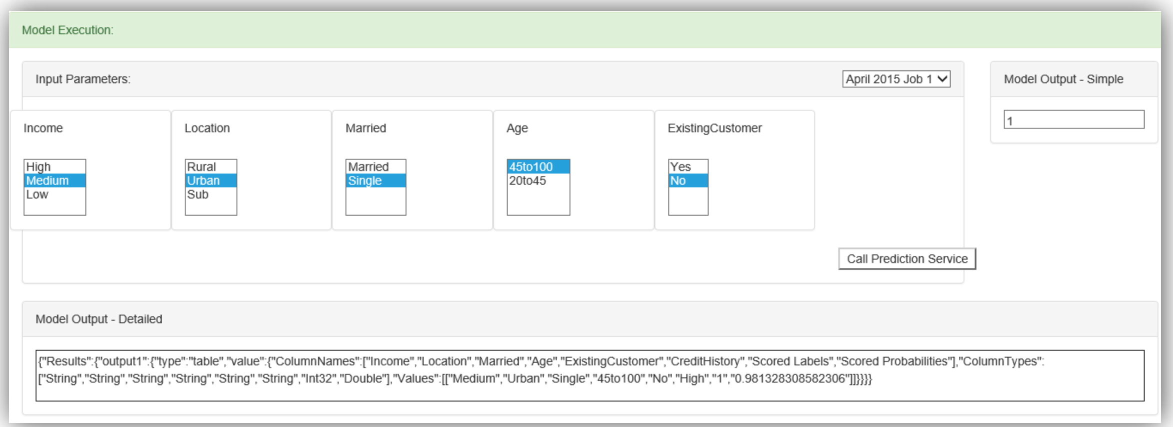


Figure 32: Model Execution Screenshot - Inputs Selected

Screenshot below shows the output of the model execution. You will notice that "Model Output –Simple" textbox shows the output in a simple format (1 in the case of this particular classification job) whereas "Model Output –Detailed" screenshot shows complete details of the input sent to the machine learning service, and the output received:



Model Execution:

Input Parameters: April 2015 Job 1

Income	Location	Married	Age	ExistingCustomer
High	Rural	Married	45to100	Yes
Medium	Urban	Single	20to45	No
Low	Sub			

Selected: Medium, Urban, Single, 45to100, No

Model Output - Simple: 1

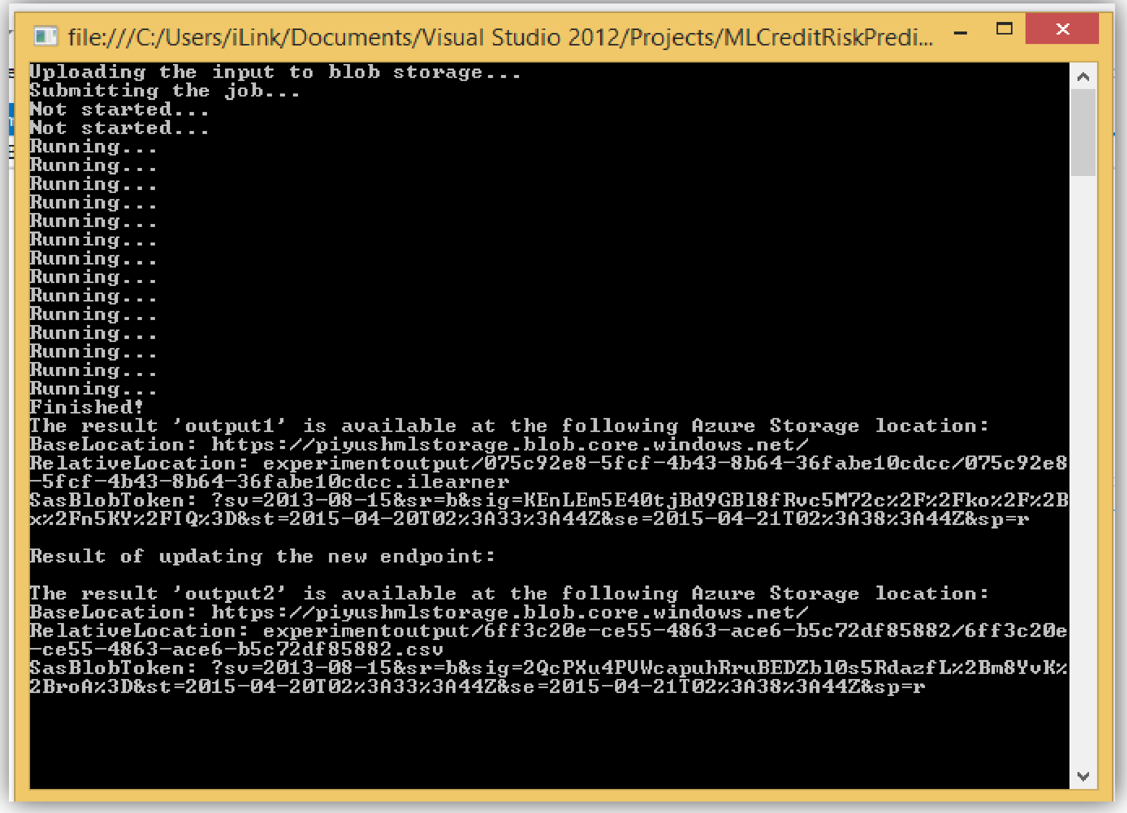
Model Output - Detailed

```
{"Results":{"output1":{"type":"table","value":{"ColumnNames":["Income","Location","Married","Age","ExistingCustomer","CreditHistory","Scored Labels","Scored Probabilities"],"ColumnTypes":
["String","String","String","String","String","String","Int32","Double"],"Values":[["Medium","Urban","Single","45to100","No","High","1","0.981328308582306"]]}}}}
```

Figure 33: Prediction Outcome

iv. Model Retrain Service:

Following screenshot shows the output from the *File Watcher Service* when executed from the console prompt:


file:///C:/Users/iLink/Documents/Visual Studio 2012/Projects/MLCreditRiskPredi...

```
Uploading the input to blob storage...
Submitting the job...
Not started...
Not started...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Running...
Finished!
The result 'output1' is available at the following Azure Storage location:
BaseLocation: https://piyushmlstorage.blob.core.windows.net/
RelativeLocation: experimentoutput/075c92e8-5fcf-4b43-8b64-36fabe10cdcc/075c92e8
-5fcf-4b43-8b64-36fabe10cdcc.ilearner
SasBlobToken: ?sv=2013-08-15&sr=b&sig=KEnLEm5E40tjBd9GB18fRvc5M72c%2F%2Fko%2F%2B
x%2Fn5KY%2FIQ%3D&st=2015-04-20T02%3A33%3A44Z&se=2015-04-21T02%3A38%3A44Z&sp=r

Result of updating the new endpoint:

The result 'output2' is available at the following Azure Storage location:
BaseLocation: https://piyushmlstorage.blob.core.windows.net/
RelativeLocation: experimentoutput/6ff3c20e-ce55-4863-ace6-b5c72df85882/6ff3c20e
-ce55-4863-ace6-b5c72df85882.csv
SasBlobToken: ?sv=2013-08-15&sr=b&sig=2QcPXu4PVWcapuhRruBEDZbl0s5RdazfL%2Bm8YvK%
2BroA%3D&st=2015-04-20T02%3A33%3A44Z&se=2015-04-21T02%3A38%3A44Z&sp=r
```

Figure 34: Model Retrain Service Output

4.2 Code Structure

Following table presents a short introduction to the item/folder in the ASP.net project structure for Predikt:

#	Name	Overview
1	References	References to .Net as well as external libraries used in the UI project.
2	Themes	Folder used to structure various CSS files used in the project, including Bootstrap css files.

Table 2

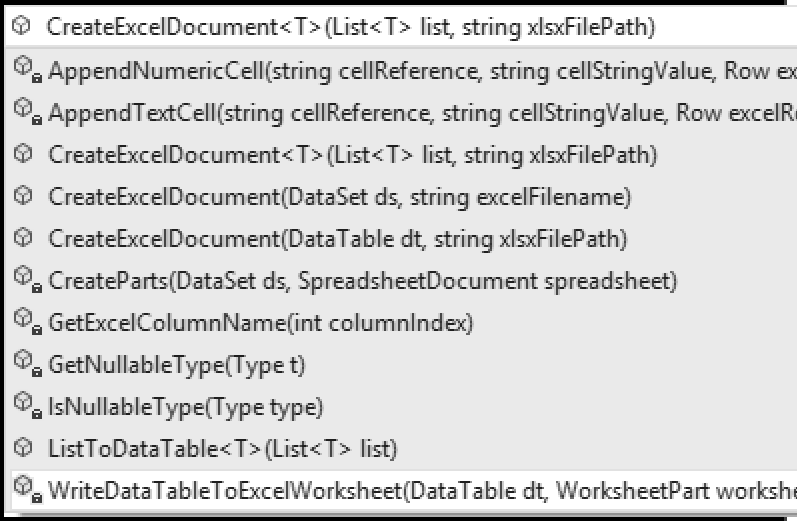
3	**Import Library**	**A set of utility classes written to assist with processing uploading CSV and excel files. Following image shows various functions that are written for this library:**  CreateExcelDocument<T>(List<T> list, string xlsxFilePath) AppendNumericCell(string cellReference, string cellStringValue, Row ex AppendTextCell(string cellReference, string cellStringValue, Row excelR CreateExcelDocument<T>(List<T> list, string xlsxFilePath) CreateExcelDocument(DataSet ds, string excelFilename) CreateExcelDocument(DataTable dt, string xlsxFilePath) CreateParts(DataSet ds, SpreadsheetDocument spreadsheet) GetExcelColumnName(int columnIndex) GetNullableType(Type t) IsNullableType(Type type) ListToDataTable<T>(List<T> list) WriteDataTableToExcelWorksheet(DataTable dt, WorksheetPart workshe
4	JS	This folder is used to organize JavaScript libraries used by ASP.net as well as a JavaScript file written to interface with Asp.net Web Services API.
5	Model	Folder used to use the Entity Framework model for Predikt.
6	Scripts	Folder used to store 3rd party JavaScript libraries used in the project including: i. JQuery 1.9 ii. Modernizr 2.6 for browser compatibility detection iii. Redips JS for drag and drop capability

Table 2 Continued

#	Name	Overview
7	UploadedDocuments & MovedDocuments	These server side folders are used to ensure that the data uploaded by users are not lost in case of a server side failure. This also ensures that uploaded data is available to be re-processed once server recovers from the failure. Uploaded files are first stored in the UploadedDocuments folder, and then moved to MovedDocuments folder on completion of the processing. In case of any failure, the file is kept in UploadedDocuments till server recovers. File is moved from the MovedDocuments folder to another folder where "RetrainService" picks it up for submission to Azure ML service. This ensures that the same file is not processed twice by the Azure ML service.
8	FileHandleerManager .ashx	This is an ASP.NET HTTP handler process that provides an endpoint to file upload control. It encapsulates the logic to create a file stream to save the uploaded file to the web server.
9	MyControls.cs	A class that is used to serialize and de-serialize UI controls used to achieve dynamic rendering of controls based on user domain.
10	Web.config	Configuration file for ASP.net project which enables environment specific variables to set Database connection string, Azure Credentials, folder names and other parameters.
11	WebForm1.aspx	Main page for the application that holds the overall layout and HTML structure for the page.
12	XMLHelper.cs	A utility class to assist with XML processing.

Table 2: ASP.Net Project Structure

4.3 Code Metrics

Following section provides Code Metrics related to the two Visual Studio projects for Predikt.

Following is a short introduction of the Metrics captured for the two projects:

i. **Maintainability Index**: Measures ease of code maintenance. Higher value is better.

ii. **Cyclomatic Complexity:** Measures number of branches. Lower values are better.

iii. **Depth of Inheritance:** Measures length of object inheritance hierarchy. Lower values are better.

iv. **Class Coupling:** Measures number of classes that are references. Lower values are better.

v. **Lines of Code:** Approximates the lines of executable code. Lower values are better.

vi. **Response Time:** The time taken for web service calls to Azure ML and Storage service.

Metrics (i) to (v) were captured using Visual Studio 2012 Code Metrics tool, whereas metric (vi) was captured using Fiddler 4.0.

a. Following table provides the code metrics for the ASP.Net project:

Maintainability Index	86
Cyclomatic Complexity	347
Depth of Inheritance	5
Class Coupling	187
Lines of Code	1054

Table 3: SE Metrics for Project 1

b. Following table provides the code metrics for the Model Retrain Service Project:

Maintainability Index	86
Cyclomatic Complexity	58
Depth of Inheritance	1
Class Coupling	31
Lines of Code	133

Table 4: SE Metrics for Project 2

c. Azure ML Web service response times: Following table presents the average response times for calls to various Azure services.

Service	Response Time (Secs)
Azure ML Model Execution Service	03.948
Azure Storage Service (uploading the new data)	00.408
Azure ML Model Training Job Submission Service	• **For 2,000 training records:** 03.092 • **For 10,000 training records:** 04.842
Azure ML Job Status Check Service	00.261
Azure ML Job Completion and Endpoint binding service	02.910

Table 5: Azure ML Web Service Response Time

The response time presented above is based on the following Azure subscription details:

- **Storage Account:**
    - Location - South Central US
    - Account type - Locally Redundant
    - Subscription type - Pay-As-You-Go
- **ML Studio Subscription:**
    - # of Web services deployed - 8
    - Location - South Central US
    - Subscription type - Pay-as-you-go
- **Model Training Web Services Endpoint:**
    - Throttle level - High
    - Diagnostic trace level - None
    - Subscription Type - Pay-As-You-Go
    - Location - South Central US

4.4 PROJECT TIMELINE

The overall time taken for the project was about 4 months with about 25% average bandwidth dedicated on an ongoing basis. Initial four weeks were spent doing literature research, finalization of scope and machine learning engine/service to be used for the project. After scope and Machine Learning Service were finalized, I also reached out to the Azure ML team at Microsoft to get answers to few open questions (as Azure ML was in beta phase and many features were not documented). With the help from the Azure ML team, I was able to successfully do a Proof of Concept to validate availability of needed services for this application. POC was followed by the development of User Interface with ASP.net 4.5 and Bootstrap 3.0.

Following table presents the key milestones:

Milestone	Duration (Weeks)
Scope Definition	2
Selection of Machine Learning Engine	2
POC with Azure ML Service	3
Design	2
UI Development	2
Integration with Azure ML	1
Testing	2
Report Creation	2
Total Duration (Weeks)	**16**

Table 6: Project Timeline

4.5 Test Cases

I created a total of **24 test cases** to cover for various combinations of data inputs and prediction requirements supported by Predikt.

The test cases that I created for this project can be categorized as follows:

- Test Cases to validate the result by changing the number of input fields.
- Test Cases to validate the result by changing the Input Field names.
- Test Cases to validate the result by providing different data types (strings, number, decimals).
- Test Cases for data pre-processing rules such as:
    a. # of bins
    b. Handling of –ve and blank values

Following are examples of some of the test cases:

i. **Test Case 1:**

- Domain: Finance – Credit Risk Prediction
- # of Input Variables: 6
- Input File Format:

Income	Location	Married	Age	Existing Customer	CreditHistory	CreditRisk
High	Rural	Married	45to100	Yes	Bad	1
Medium	Urban	Single	45to100	No	Good	1
Low	Sub	Married	45to100	Yes	Ugly	1
Low	Urban	Single	20to45	No	Ugly	0

Table 7: Test Case 1 Input

- Expected UI for prediction:

Text Area	1	2	3	4	5	6
Heading	Income	Location	Married	Age	Existing Customer	Credit History
Values	High Medium Low	Rural Urban Sub	Married Single	20to45 45to100	Yes No	Good Bad Ugly

Table 8: Test Case 1 Output

ii. **Test Case 2:**

- Domain: Oil & Gas – ESP Failure Prediction
- # of Input Variables: 5
- Input File Format:

WellType	Drill Type	RockType	Basin	Hours	FailureRisk
Oil	Horizontal	Clay	Permian	357	1
Gas	Vertical	Shale	Eagleford	155	1
Water	Horizontal	Coal	Saudi	941	1
Gas	Vertical	Shale	Kazakhstan	32	0

Table 9: Test Case 2 Input

- Expected UI for prediction:

Text Area	1	2	3	4	5
Heading	**WellType**	**Drill Type**	**RockType**	**Basin**	**Hours**
Values	Oil Gas Water	Horizontal Vertical	Clay Shale Coal	Permian Eagleford Saudi Kazakhstan	4 bins

Table 10: Test Case 2 Output

Chapter 5: Conclusion

Machine Learning is a critical piece of how Big Data is harnessed. Implementation of Predikt was an attempt to simplify the machine learning experience to the extent where it can be adopted by the masses with or without any knowledge of underlying mathematical complexity. I started this project by stating out the guiding principles and documenting the detailed requirements. Design phase helped in solidifying the User Interface as well as technical design for the project, followed by implementation and testing of the application.

Below I conclude by presenting various lessons learnt and things that went well. I have also presented few items in the *Future Work* section that I believe are needed in Predikt for it realize it's full potential, and to create a comprehensive set of features that can be taken to market.

5.1 Lessons Learnt:

Lessons learnt can be categorized into two sections:

a. Technology
b. Project management

Lessons learnt in each of these areas are presented below:

a. Technology:

i. Server Side Vs HTML Control:

It is important to finalize the choice of controls to be used in user interface implementation during the design phase. When I started the implementation phase, I had a rough idea on what controls to use. I picked the UI control as project progressed based on the specific requirements. Late into a project I realized that I had ended up using a combination of ASP.Net Server side controls and client side HTML controls to create the UI. It was getting difficult for them to talk to each other, which led to significant rework. For example, The identifier for client side controls was not readily accessible from event handler for Server side controls, and I was having to find workaround to make it work.

Following table presents a mapping for some of the common ASP.net Server side and HTML controls.

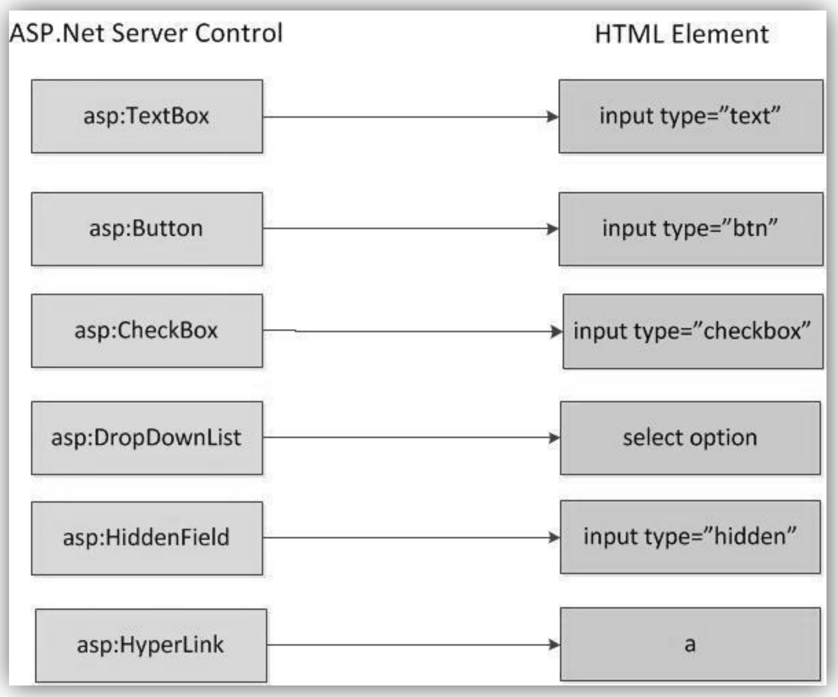


Figure 35: Server Side and HTML Control Mapping

ii. **Choice of Machine Learning Engine:**

One of the early decisions that I had to do make was identification of the right machine learning engine for Predikt. Although I had to try multiple options before zeroing in on Azure ML service, I was caught unprepared because Azure ML Service is in a very early stage. Although the key factor into picking Azure ML service was availability of web services to interact with Machine Learning models, the fact that these web services were not fully documented made it difficult to use them. I spent significant amount of time trying to figure it out myself, before reaching out to Microsoft Support team, who were very helpful. Most of the time they were able to respond back to my queries within a day with proper (many times unpublished) documentation. Reaching out to them early on would have saved significant time and effort.

Also, given that Azure was in an early stage of release, there were operational issues such as I was not able to move my models from the free tier to paid tier. I had to recreate all my models in the paid tier, which could have been avoided if this was a known limitation.

iii. **Asynchronous Processing:** Predikt needs to support concurrent data upload and model re-training. Given that I am using Shared Folder to store the uploaded file before pre-processing rules are run, and then it is uploaded to Azure blob for model training. I ran into several issues because of File Watcher Service was not designed to process multiple files at a time. Similarly, Azure web services calls were not designed to be asynchronous which led to the program control being stuck till the response comes back. I had to rework the code to use .net `System.Threading` namespace to ensure that uploaded files are processed in parallel, and that all calls to Azure web services are also not done asynchronously.

b. Project Management:

i. Cost Control:

Although Azure Ml service provided a streamlined way for me to access various Machine Learning models, it also was costing me money for every transaction and every bit of data that I was storing in Azure. I was surprised by how quickly the amount adds up. The total spend for this project was few hundred dollars. Although the amount in this instance is not very high, but it is making me question the long term impact of choice between 3rd party service hosted model (SaaS) Vs self-hosted model. This is definitely something that I will need to re-consider as I plan to take Predikt to market.

ii. Scope Definition:

I did not document the features needed in Predikt which led to last minute scrambling to meet the expectations. The invaluable lesson learnt is to understand the stakeholder expectations early on and document via wireframes to ensure that there is a shared understanding of overall scope. Additionally, I was weighing in multiple choices for this exercise (master's project). Delay in making decision about the high level scope led to delay in subsequent phases as well.

iii. Milestones:

Lack of milestone definitions also led to inadequate tracking of project completeness. Having intermittent milestones would have helped catch any delay as well as validation of the expectations with respect to scope as the project progressed.

iv. Testing:

I underestimated the effort required to test multiple combinations of scenarios that Predikt is expected to support. Issue was intensified because of need to re-test all the scenarios multiple times after any change in code. Planning for QA automation would have prevented this issue from happening.

5.2 What Went Right

1. **Appreciation for the Machine Learning domain:**

   Going through this exercise and diving into the complexity of machine learning domain has given me a deep appreciation for the potential as well as risks associated with decision making using these techniques.

2. **Cloud based Machine Learning Service:**

   Although Azure ML Service is in an early stage, it provides a very streamlined way for users to interact with machine learning models. Once I was able to get the proper documentation from Microsoft Support team, the service itself worked flawlessly.

The time saving that otherwise would go into setting up infrastructure and dealing with system level issues was critical to be able to complete this project on time.

3. **Feedback from Supervisor:**

   Feedback from Dr Aziz and his insistence on meeting certain level of academic rigor with respect to this project led to a more complete solution, that I believe is at a point where it can be demonstrated to potential customers and partners.

5.3 Future Work

Following are some of the potential addition needed in Predikt to make it into a comprehensive solution ready for market:

a. **Role Based Access:** For Predikt to be a hosted service supporting multiple customers and domains, having a secure role based access implementation is critical. Because data for multiple customers is going to reside in the same database, special consideration needs to be given to ensure that data is properly secured and not accessible across customer boundaries. Although database is designed to supported multiple users, UI and service layers need to be enhanced to provide this feature.

b. **More Control over Model Training and Execution:** More control can be provided in the form of data pre-processing rules such as column level pre-processing rules. For example, currently number of bins specified by users gets applied to all the numeric columns.

c. **Support for multiple levels of licensing:** Screens can be provided for end users to select the number of models that they want to be trained and evaluated for their dataset. A licensing regimen can be defined based upon number of models that user selects (and the volume of their data).

d. **Support for more machine learning models:** Currently only three classification models are enabled (Support Vector Machine, Boosted Decision Tree and Logistic Regression). More machine learning models can be added for Predikt to be a comprehensive self-service data mining service.

e. **Incorporate more real-time data sources:** The basic premise of being able to predict future based on what happened in the past can lead to incorrect decision making. Incorporating latest data outside of customer's domain (such as current Macro economic data and Social trends) into decision making can add significant value to customer.

Bibliography

1. Magoulas, Roger; Lorica, Ben (February 2009). "Introduction to Big Data". Release 2.0 (Sebastopol CA: O’Reilly Media) (11).

2. Snijders, C.; Matzat, U.; Reips, U.-D. (2012). "'Big Data': Big gaps of knowledge in the field of Internet". International Journal of Internet Science 7: 1–5.

3. Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, Samee Ullah Khan, The rise of "big data" on cloud computing: Review and open research issues, Information Systems, Volume 47, January 2015, Pages 98-115, ISSN 0306-4379, http://dx.doi.org/10.1016/j.is.2014.07.006

4. Big Data Computing and Clouds: Challenges, Solutions, and Future Directions. Marcos D. Assuncao, Rodrigo N. Calheiros, Silvia Bianchi, Marco A. S. Netto, Rajkumar Buyya. Technical Report CLOUDS-TR-2013-1, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne, 17 Dec. 2013.

5. Thomas H. Davenport and D.J. Patil. Data scientist: The sexiest job of the 21st century. Harvard Business Review, October 2012. Available: http://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century/ .

6. Constance L. Hays. What they know about you. The New York Times, Nov 14 2004.

7. Consumer Credit Risk Models via Machine-Learning Algorithms Amir E. Khandaniy, Adlar J. Kimz, and Andrew W. Lox [MIT Sloan School of Management]

8. Data Mining Problems in Retail: https://highlyscalable.wordpress.com/2015/03/10/data-mining-problems-in-retail/

9. What is data mining? http://docs.oracle.com/cd/B28359_01/datamine.111/b28129/process.htm

10. Rokach, Lior; Maimon, O. (2008). Data mining with decision trees: theory and applications. World Scientific Pub Co Inc. ISBN 978-9812771711.

11. Quinlan, J. R., (1986). Induction of Decision Trees. Machine Learning 1: 81-106, Kluwer Academic Publishers

12. Breiman, Leo; Friedman, J. H.; Olshen, R. A.; Stone, C. J. (1984). Classification and regression trees. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software. ISBN 978-0-412-04841-8.

13. Boyd, D.; Crawford, K. (2012). "Critical Questions for Big Data". Information, Communication & Society 15 (5): 662. doi:10.1080/1369118X.2012.678878.

14. Ron Kohavi; Foster Provost (1998). "Glossary of terms". Machine Learning 30: 271–274.

15. C. M. Bishop (2006). Pattern Recognition and Machine Learning. Springer. ISBN 0-387-31073-8.

16. Wernick, Yang, Brankov, Yourganov and Strother, Machine Learning in Medical Imaging, IEEE Signal Processing Magazine, vol. 27, no. 4, July 2010, pp. 25-38

17. Mannila, Heikki (1996). Data mining: machine learning, statistics, and databases. Int'l Conf. Scientific and Statistical Database Management. IEEE Computer Society

Vita

Piyush has over 15 years of experience working on an array of strategic planning, technology assessment, internet infrastructure, software architecture, and development and implementation projects. He has worked with several Fortune 500 clients as well as startups and product companies on a variety of Software Engineering Projects. Piyush has managed large teams as well as developed and delivered enterprise solutions. His experience spans several industries including Oil & Gas, HealthCare, Finance, Chemical and Information Technology.

Permanent address: Prakash.piyush@gmail.com

13903 Briar Place Dr,

Houston, TX 77077

This report was typed by Piyush Prakash.