Data Mining Comparison
SPSS Modeler vs Spark_Python

David Douglas
Rekha Swaminathan
Department of Information Systems
Thank you
IBM
Data Mining Methodology

Cross Industry Standard Process – Data Mining

SAS Methodology
S-Sample
E-Explore
M-Modify
M-Model
A-Assess

CRISP-DM
Characteristics of Good Predictive Models

- Preventing Overfitting for Predictive Models
  - Training sample for building the model
  - Critical to have a hold-out sample to determine if the model has overfitted the data
  - Compare model built with training data to model build with hold-out sample
    - Classification models; misclassification rate, confusion table, gains (lift) charts, ROC charts
    - Estimation models; numeric values such as RMSE

- Perhaps 60% of effort is data preparation
  - Domain knowledge, removing bad data, adding relevant data, dimension reduction, ...
Comparison Plan

IBM SPSS Modeler (Client and Server) compared to IBM Data Science with Spark/Python

- Classification models – Decision Trees (DT) and Neural Networks (NN)
- Telco churn dataset – 3333 records
  - Data exploration/modification
  - Model runs and assessment
- Medium Dillard’s dataset predicting returns – 1 million records
- Larger Dillard’s Datasets – many millions records in a database table using database connection features of SPSS Modeler and Data Science data connection
Accomplished

IBM SPSS Modeler (Client and Server) compared to IBM Data Science with Spark/Python

• Classification models – Decision Trees (DT) and Neural Networks (NN)
• Telco churn dataset – 3333 records
  • Data exploration/modification – limited Data Science
  • Model runs and assessment
• Medium Dillard’s dataset predicting returns – 1 million records
• Larger Dillard’s Datasets – many millions records in a database table using database connection features of SPSS Modeler and Data Science data connection
• Never received connector but uploaded 13 million row flat file to Data Science—neither DT nor NN completed in 5 hours. SPSS Modeler completed DT in 1 hr but not NN
Data for Building the Model
Comparisons

• Predictive models using Decision Trees and Neural Nets
• Small telco dataset – Churn
  – 3333 records consisting of 20 predictors and 1 target
  – Target is Churn? which indicates if customer left the company or not and has values of True/False
  – State, area code, phone, and charges (day, evening, night, international) removed because of various reasons.
• Records from Dillard’s dataset
  – 7 predictors and 1 target
  – All predictors retained, target is Trans_Type with values of P (Purchase) or R(return)
Features Comparison

IBM SPSS Modeler
- Model flow driven generally requiring no programming
- Extensive exploration/data modification scripting
- Takes care of little things like automatically coding categorical data
- Extensive built in modeling techniques – for example 24 classification models
- Auto features – for example auto classifier
- Automatic output—classification tables, ROC charts, etc.
- Extensive data access including big data – Hadoop
- Coding nodes for R and Python if needed

IBM Data Science
- Open Source – e.g. Spark, Python/R/Stata, etc.
- Extensive machine learning libraries; however, may need to use different libraries depending on model
- Requires multiple linear runs to get to the answer
- Requires extensive programming skills
- Have to code categorical values to numeric indicator variables
- Generally have to work hard to get output in nice format
- Should be able to model larger datasets but that is not what we encountered
Decision Trees and Neural Networks using IBM SPSS Modeler
Decision Tree and Neural Network Model Flow with Modeler
# Data with Modeler

The original data to be used for both Modeler and Python

**Notes:**
- The target variables is Churn? with False. and True. as valid entries
- Several of the predictor variables are categorical
- The data contains a number of data issues
- The data can be used “as is” with Modeler

<table>
<thead>
<tr>
<th></th>
<th>State</th>
<th>Account Length</th>
<th>Area Code</th>
<th>Phone</th>
<th>Intl Plan</th>
<th>VMail Plan</th>
<th>VMail Msg</th>
<th>Day Calls</th>
<th>Day Mins</th>
<th>Day Charge</th>
<th>Eve Mins</th>
<th>Eve Calls</th>
<th>Eve Charge</th>
<th>Night Mins</th>
<th>Night Calls</th>
<th>Night Charge</th>
<th>Intl Mins</th>
<th>Intl Calls</th>
<th>Intl Charge</th>
<th>CustServ Calls</th>
<th>Churn?</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>OH</td>
<td>84</td>
<td>408</td>
<td>375-9999</td>
<td>yes</td>
<td>no</td>
<td>0</td>
<td>299.4</td>
<td>71</td>
<td>50.9</td>
<td>61.9</td>
<td>88</td>
<td>5.26</td>
<td>196.9</td>
<td>89</td>
<td>8.86</td>
<td>6.6</td>
<td>7</td>
<td>1.78</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>LA</td>
<td>117</td>
<td>408</td>
<td>335-4719</td>
<td>no</td>
<td>no</td>
<td>0</td>
<td>184.5</td>
<td>97</td>
<td>31.37</td>
<td>351.6</td>
<td>80</td>
<td>29.89</td>
<td>215.8</td>
<td>90</td>
<td>9.71</td>
<td>8.7</td>
<td>4</td>
<td>2.35</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>IA</td>
<td>168</td>
<td>408</td>
<td>363-1107</td>
<td>no</td>
<td>no</td>
<td>0</td>
<td>126.8</td>
<td>96</td>
<td>21.9</td>
<td>104.9</td>
<td>71</td>
<td>8.92</td>
<td>141.1</td>
<td>128</td>
<td>6.35</td>
<td>11.2</td>
<td>2</td>
<td>3.02</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ID</td>
<td>85</td>
<td>408</td>
<td>350-8884</td>
<td>yes</td>
<td>yes</td>
<td>27</td>
<td>196.4</td>
<td>139</td>
<td>33.39</td>
<td>280.9</td>
<td>90</td>
<td>23.88</td>
<td>89.3</td>
<td>75</td>
<td>4.02</td>
<td>13.8</td>
<td>4</td>
<td>3.73</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>CO</td>
<td>77</td>
<td>408</td>
<td>393-7984</td>
<td>no</td>
<td>no</td>
<td>0</td>
<td>62.4</td>
<td>89</td>
<td>10.61</td>
<td>169.9</td>
<td>121</td>
<td>14.44</td>
<td>209.6</td>
<td>84</td>
<td>9.43</td>
<td>5.7</td>
<td>6</td>
<td>1.54</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>WY</td>
<td>57</td>
<td>408</td>
<td>357-3817</td>
<td>yes</td>
<td>yes</td>
<td>39</td>
<td>213</td>
<td>115</td>
<td>36.21</td>
<td>191.1</td>
<td>112</td>
<td>162.4</td>
<td>182.7</td>
<td>115</td>
<td>8.22</td>
<td>9.5</td>
<td>3</td>
<td>2.57</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>MT</td>
<td>54</td>
<td>408</td>
<td>418-8412</td>
<td>no</td>
<td>no</td>
<td>0</td>
<td>134.3</td>
<td>73</td>
<td>22.83</td>
<td>155.5</td>
<td>100</td>
<td>13.22</td>
<td>102.1</td>
<td>68</td>
<td>4.59</td>
<td>14.7</td>
<td>4</td>
<td>3.97</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>LA</td>
<td>172</td>
<td>408</td>
<td>383-1121</td>
<td>no</td>
<td>no</td>
<td>0</td>
<td>212</td>
<td>121</td>
<td>36.04</td>
<td>31.2</td>
<td>115</td>
<td>2.65</td>
<td>293.3</td>
<td>78</td>
<td>13.2</td>
<td>12.6</td>
<td>10</td>
<td>3.4</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>AZ</td>
<td>12</td>
<td>408</td>
<td>360-1596</td>
<td>no</td>
<td>no</td>
<td>0</td>
<td>249.6</td>
<td>118</td>
<td>42.43</td>
<td>252.4</td>
<td>119</td>
<td>21.45</td>
<td>280.2</td>
<td>90</td>
<td>12.61</td>
<td>11.8</td>
<td>3</td>
<td>3.19</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>OK</td>
<td>57</td>
<td>408</td>
<td>395-2854</td>
<td>yes</td>
<td>yes</td>
<td>25</td>
<td>176.8</td>
<td>94</td>
<td>30.06</td>
<td>195</td>
<td>75</td>
<td>16.58</td>
<td>213.5</td>
<td>116</td>
<td>9.61</td>
<td>8.3</td>
<td>4</td>
<td>2.24</td>
<td>False</td>
<td></td>
</tr>
</tbody>
</table>
### Modeler Data Types Window

<table>
<thead>
<tr>
<th>Field</th>
<th>Measurement</th>
<th>Values</th>
<th>Missing</th>
<th>Check</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>Nominal</td>
<td>AK, AL, AR, AZ, CA, CO, CT, DC, DE, FL, GA, HI, IA, ID, IL, IN, KS, KY, LA, ME, MI, MN, MO, MS, MT, NE, NV, NH, NJ, NM, NY, OH, OK, OR, PA, RI, SC, SD, TN, TX, UT, VA, VT, WA, WI, WV, WY</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Account Length</td>
<td>Continuous</td>
<td>[1.0, 2.43]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Area Code</td>
<td>Nominal</td>
<td>408, 615, 510</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Phone</td>
<td>Typeless</td>
<td></td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>IntlPlan</td>
<td>Flag</td>
<td>yes/no</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>VMail Plan</td>
<td>Flag</td>
<td>yes/no</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>VMail Message</td>
<td>Continuous</td>
<td>[0.0, 0.51]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Day Mins</td>
<td>Continuous</td>
<td>[0.0, 0.350]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Day Calls</td>
<td>Continuous</td>
<td>[0.0, 1.65]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Day Charge</td>
<td>Continuous</td>
<td>[0.0, 0.5964]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Eve Mins</td>
<td>Continuous</td>
<td>[0.0, 0.363]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Eve Calls</td>
<td>Continuous</td>
<td>[0.0, 0.17]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Eve Charge</td>
<td>Continuous</td>
<td>[0.0, 0.3091]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Night Mins</td>
<td>Continuous</td>
<td>[23.2, 2395]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Night Calls</td>
<td>Continuous</td>
<td>[33.0, 175]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Night Charge</td>
<td>Continuous</td>
<td>[1.04, 17.77]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Intl Mins</td>
<td>Continuous</td>
<td>[0.0, 2.0]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Intl Calls</td>
<td>Continuous</td>
<td>[0.0, 2.0]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Intl Charge</td>
<td>Continuous</td>
<td>[0.0, 5.4]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>CustServ Calls</td>
<td>Continuous</td>
<td>[0.0, 9.0]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
</tbody>
</table>
| Churn?         | Flag        | True/False      | None    | None        | Target
## Data Audit Node

### Data Audit of [20 fields] #1

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample Graph</th>
<th>Measurement</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Skewness</th>
<th>Unique</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State</strong></td>
<td></td>
<td>Nominal</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>51</td>
<td>3333</td>
</tr>
<tr>
<td><strong>Account Len...</strong></td>
<td></td>
<td>Continuous</td>
<td>1.000</td>
<td>243.000</td>
<td>101.065</td>
<td>39.822</td>
<td>0.097</td>
<td>--</td>
<td>3333</td>
</tr>
<tr>
<td><strong>Area Code</strong></td>
<td></td>
<td>Nominal</td>
<td>408.000</td>
<td>510.000</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>3</td>
<td>3333</td>
</tr>
<tr>
<td><strong>InitPlan</strong></td>
<td></td>
<td>Flag</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>2</td>
<td>3333</td>
</tr>
</tbody>
</table>
Strong Correlations – Statistics Node

The image shows a statistical node with the statistics of various fields. The highlighted row is for "Day Mins" with the following statistics:

- Count: 3333
- Mean: 179.775
- Min: 0.000
- Max: 350.800
- Range: 350.800
- Variance: 2966.696
- Standard Deviation: 54.467
- Standard Error of Mean: 0.943

The Pearson Correlations section highlights "Day Charge" with a correlation coefficient of 1.000, indicating a strong positive correlation.
# Data Settings for Running the Model

<table>
<thead>
<tr>
<th>Field</th>
<th>Measurement</th>
<th>Values</th>
<th>Missing</th>
<th>Check</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>Nominal</td>
<td>AK, AL, AR, AZ, CA, CO, ...</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Account Length</td>
<td>Continuous</td>
<td>[1.0, 243.0]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Area Code</td>
<td>Nominal</td>
<td>408.0, 415.0, 510.0</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Phone</td>
<td>Typeless</td>
<td></td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>IntlPlan</td>
<td>Flag</td>
<td>yes/no</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>VMail Plan</td>
<td>Flag</td>
<td>yes/no</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>VMail Message</td>
<td>Continuous</td>
<td>[0.0, 51.0]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Day Mins</td>
<td>Continuous</td>
<td>[0.0, 350.8]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Day Calls</td>
<td>Continuous</td>
<td>[0.0, 165.0]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Day Charge</td>
<td>Continuous</td>
<td>[0.0, 59.64]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Eve Mins</td>
<td>Continuous</td>
<td>[0.0, 363.7]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Eve Calls</td>
<td>Continuous</td>
<td>[0.0, 170.0]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Eve Charge</td>
<td>Continuous</td>
<td>[0.0, 30.91]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Night Mins</td>
<td>Continuous</td>
<td>[23.2, 395.0]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Night Calls</td>
<td>Continuous</td>
<td>[33.0, 175.0]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Night Charge</td>
<td>Continuous</td>
<td>[1.04, 17.77]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Intl Mins</td>
<td>Continuous</td>
<td>[0.0, 20.0]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Intl Calls</td>
<td>Continuous</td>
<td>[0.0, 20.0]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Intl Charge</td>
<td>Continuous</td>
<td>[0.0, 5.4]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>CustServ Calls</td>
<td>Continuous</td>
<td>[0.0, 9.0]</td>
<td>None</td>
<td>None</td>
<td>Input</td>
</tr>
<tr>
<td>Churn?</td>
<td>Flag</td>
<td>True/False</td>
<td>None</td>
<td>None</td>
<td>Target</td>
</tr>
</tbody>
</table>
Decision Tree
Output – Churn
1.89 seconds
Portion of Decision Tree
English Rules of Decision Tree

CustServ Calls <= 3 [ Mode: False. ] (1,762)
- Day Mins <= 224.100 [ Mode: False. ] (1,393)
  - IntlPlan = yes [ Mode: False. ] (119)
    - Intl Calls <= 2 [ Mode: True. ] \implies True. (19, 1.0)
    - Intl Calls > 2 [ Mode: False. ] (100)
      - Intl Mins <= 13.100 [ Mode: False. ] \implies False. (82, 1.0)
      - Intl Mins > 13.100 [ Mode: True. ] \implies True. (18, 1.0)
  - IntlPlan = no [ Mode: False. ] \implies False. (1,274, 0.976)

- Day Mins > 224.100 [ Mode: False. ] (369)
  - VMail Plan = yes [ Mode: False. ] \implies False. (92, 0.935)
  - VMail Plan = no [ Mode: False. ] (277)
    - Eve Mins <= 266.900 [ Mode: False. ] (246)
    - Day Mins <= 276.200 [ Mode: False. ] (206)
      - Eve Mins <= 202.500 [ Mode: False. ] (114)
        - IntlPlan = yes [ Mode: False. ] (17)
        - Intl Calls <= 2 [ Mode: True. ] \implies True. (6, 1.0)
        - Intl Calls > 2 [ Mode: False. ] \implies False. (11, 1.0)
        - IntlPlan = no [ Mode: False. ] \implies False. (97, 0.928)
### Classification Table

#### Results for output field Churn?

<table>
<thead>
<tr>
<th>Individual Models</th>
<th>Comparing $C$-Churn? with Churn?</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Partition'</td>
<td>1_Training</td>
</tr>
<tr>
<td></td>
<td>2_Testing</td>
</tr>
<tr>
<td>Correct</td>
<td>1,850</td>
</tr>
<tr>
<td></td>
<td>96.61%</td>
</tr>
<tr>
<td></td>
<td>1,330</td>
</tr>
<tr>
<td></td>
<td>93.79%</td>
</tr>
<tr>
<td>Wrong</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>3.39%</td>
</tr>
<tr>
<td></td>
<td>88</td>
</tr>
<tr>
<td></td>
<td>6.21%</td>
</tr>
<tr>
<td>Total</td>
<td>1,915</td>
</tr>
<tr>
<td></td>
<td>1,418</td>
</tr>
</tbody>
</table>

#### Coincidence Matrix for $C$-Churn? (rows show actuals)

<table>
<thead>
<tr>
<th>'Partition' = 1_Training</th>
<th>False.</th>
<th>True.</th>
</tr>
</thead>
<tbody>
<tr>
<td>False.</td>
<td>1,645</td>
<td>3</td>
</tr>
<tr>
<td>True.</td>
<td>62</td>
<td>205</td>
</tr>
<tr>
<td>'Partition' = 2_Testing</td>
<td>False.</td>
<td>True.</td>
</tr>
<tr>
<td>False.</td>
<td>1,184</td>
<td>18</td>
</tr>
<tr>
<td>True.</td>
<td>70</td>
<td>146</td>
</tr>
</tbody>
</table>

#### Evaluation Metrics

<table>
<thead>
<tr>
<th>'Partition'</th>
<th>1_Training</th>
<th>2_Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>AUC</td>
<td>Gini</td>
</tr>
<tr>
<td>$C$-Churn?</td>
<td>0.926</td>
<td>0.853</td>
</tr>
<tr>
<td></td>
<td>0.876</td>
<td>0.753</td>
</tr>
</tbody>
</table>
ROC Chart

Evaluation of [S-C-Churn?] : ROC

Graph
Annotations

TP Rate (Sensitivity)
0.0 0.2 0.4 0.6 0.8 1.0
0.0 0.2 0.4 0.6 0.8 1.0
FP Rate (1- Specificity)
Training
Testing

Partition
'Churn?' = "True."

$S$-C-Churn?
Neural Network Output – Churn 2.08 Seconds
## Model Summary

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Target</strong></td>
<td>Churn?</td>
</tr>
<tr>
<td><strong>Model</strong></td>
<td>Multilayer Perceptron</td>
</tr>
<tr>
<td><strong>Stopping Rule Used</strong></td>
<td>Error cannot be further decreased</td>
</tr>
<tr>
<td><strong>Hidden Layer 1 Neurons</strong></td>
<td>6</td>
</tr>
</tbody>
</table>

![Accuracy Chart](image)
Predictor Importance

Target: Churn?

- Day Mins
- CusServ Calls
- Eve Mins
- Day Calls
- Intl Mins
- Night Mins
- IntlPlan
- Night Calls
- VMail Message
- Eve Calls
**Classification Table**

---

**Individual Models**

Comparing $N$-Churn? with Churn?

<table>
<thead>
<tr>
<th>'Partition'</th>
<th>1_Training</th>
<th>2_Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct</td>
<td>1,796</td>
<td>1,292</td>
</tr>
<tr>
<td></td>
<td>93.79%</td>
<td>91.11%</td>
</tr>
<tr>
<td>Wrong</td>
<td>119</td>
<td>126</td>
</tr>
<tr>
<td></td>
<td>6.21%</td>
<td>8.89%</td>
</tr>
<tr>
<td>Total</td>
<td>1,915</td>
<td>1,418</td>
</tr>
</tbody>
</table>

Coincidence Matrix for $N$-Churn? (rows show actuals)

<table>
<thead>
<tr>
<th>'Partition' = 1_Training</th>
<th>False.</th>
<th>True.</th>
</tr>
</thead>
<tbody>
<tr>
<td>False.</td>
<td>1,618</td>
<td>30</td>
</tr>
<tr>
<td>True.</td>
<td>89</td>
<td>178</td>
</tr>
<tr>
<td>'Partition' = 2_Testing</td>
<td>False.</td>
<td>True.</td>
</tr>
<tr>
<td>False.</td>
<td>1,165</td>
<td>37</td>
</tr>
<tr>
<td>True.</td>
<td>89</td>
<td>127</td>
</tr>
</tbody>
</table>

Evaluation Metrics

<table>
<thead>
<tr>
<th>'Partition'</th>
<th>1_Training</th>
<th>2_Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>AUC</td>
<td>Gini</td>
</tr>
<tr>
<td>$N$-Churn?</td>
<td>0.93</td>
<td>0.86</td>
</tr>
<tr>
<td></td>
<td>0.889</td>
<td>0.777</td>
</tr>
</tbody>
</table>
ROC Chart
The Neural Network
Predicting Returns - Dillard's Data

1 million records

- CHAID Decision Tree (14.28 Seconds)
- Neural Network (7 minutes, 28 seconds)
Same model flow as with the churn data without the Data Audit and Statistics nodes and substituting the Dillard’s data for the churn data
### Classification Table

#### Results for output field Tran_Type

**Individual Models**

- Comparing $R$-Tran_Type with Tran_Type

<table>
<thead>
<tr>
<th>'Partition'</th>
<th>1_Training</th>
<th></th>
<th>2_Testing</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Correct</strong></td>
<td>581,294</td>
<td>96.84%</td>
<td>387,180</td>
<td>96.85%</td>
</tr>
<tr>
<td><strong>Wrong</strong></td>
<td>18,950</td>
<td>3.16%</td>
<td>12,575</td>
<td>3.15%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>600,244</td>
<td></td>
<td>399,755</td>
<td></td>
</tr>
</tbody>
</table>

#### Coincidence Matrix for $R$-Tran_Type (rows show actuals)

<table>
<thead>
<tr>
<th>'Partition'</th>
<th>1_Training</th>
<th></th>
<th>2_Testing</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>R</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'Partition'</td>
<td>1_Training</td>
<td></td>
<td>2_Testing</td>
<td></td>
</tr>
<tr>
<td><strong>P</strong></td>
<td>287,919</td>
<td>12,463</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>R</strong></td>
<td>6,487</td>
<td>293,375</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Evaluation Metrics

<table>
<thead>
<tr>
<th>'Partition'</th>
<th>1_Training</th>
<th></th>
<th>2_Testing</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model</strong></td>
<td>AUC</td>
<td>Gini</td>
<td>AUC</td>
<td>Gini</td>
</tr>
<tr>
<td>$R$-Tran_Type</td>
<td>0.989</td>
<td>0.979</td>
<td>0.989</td>
<td>0.978</td>
</tr>
</tbody>
</table>
ROC Chart

Evaluation of [SR-Tran_Type] : Gains

- Graph
- Annotations

Partition
Tran_Type = "R"

% Gain
0 20 40 60 80 100
0 20 40 60 80 100

Percentile Training
Percentile Testing

$R$-Tran_Type
## Neural Network Accuracy

### Model Summary

<table>
<thead>
<tr>
<th>Target</th>
<th>Tran_Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Multilayer Perceptron</td>
</tr>
<tr>
<td>Stopping Rule Used</td>
<td>Error cannot be further decreased</td>
</tr>
<tr>
<td>Hidden Layer 1 Neurons</td>
<td>11</td>
</tr>
</tbody>
</table>

![Accuracy Chart]

The chart shows that the accuracy of the model is 98.7%.
Classification for Tran_Type
Overall Percent Correct = 98.7%

<table>
<thead>
<tr>
<th>Observed</th>
<th>Predicted</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>P</td>
<td>100.0%</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>2.7%</td>
<td>97.3%</td>
<td></td>
</tr>
</tbody>
</table>

Row Percent
- 100.00
- 80.00
- 60.00
- 40.00
- 20.00
- 0.00
ROC Chart

Partition
Tran_Type = "R"
The Neural Network
Building a Decision Tree Spark using Python on IBM Data Science (previously Bluemix)
### Churn Data Set

- **22 Columns**
- **3333 Rows**
- **Target Variable:** Churn – True/False

<table>
<thead>
<tr>
<th>ID</th>
<th>State</th>
<th>Account Len Area Code</th>
<th>Phone</th>
<th>IntPlan</th>
<th>VM Call Plan</th>
<th>VM Call Min</th>
<th>VM Call Max</th>
<th>Day Calls</th>
<th>Day Charge Eve Min</th>
<th>Eve Calls</th>
<th>Eve Charge</th>
<th>Night Calls</th>
<th>Night Charge</th>
<th>Int Mins</th>
<th>Int Calls</th>
<th>Int Charge</th>
<th>Cust/Serv Calls</th>
<th>Churn</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>KS</td>
<td>128 382-4857</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>25</td>
<td>265.1</td>
<td>110</td>
<td>45.07</td>
<td>197.4</td>
<td>99</td>
<td>16.78</td>
<td>244.7</td>
<td>91</td>
<td>11.01</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>CH</td>
<td>107 371-7191</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>20</td>
<td>161.0</td>
<td>123</td>
<td>27.47</td>
<td>195.5</td>
<td>103</td>
<td>16.62</td>
<td>254.4</td>
<td>103</td>
<td>11.45</td>
<td>13.7</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>NJ</td>
<td>137 358-1921</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>0</td>
<td>243.4</td>
<td>114</td>
<td>41.38</td>
<td>121.2</td>
<td>110</td>
<td>10.3</td>
<td>162.8</td>
<td>104</td>
<td>7.32</td>
<td>12.2</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>CH</td>
<td>107 375-9999</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>0</td>
<td>259.1</td>
<td>71</td>
<td>50.9</td>
<td>61.9</td>
<td>68</td>
<td>5.26</td>
<td>156.3</td>
<td>66</td>
<td>8.86</td>
<td>6.6</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>OK</td>
<td>117 330-8626</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>0</td>
<td>166.7</td>
<td>113</td>
<td>28.34</td>
<td>148.3</td>
<td>122</td>
<td>12.61</td>
<td>186.9</td>
<td>121</td>
<td>8.41</td>
<td>16.1</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>AL</td>
<td>118 391-0227</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>0</td>
<td>223.4</td>
<td>98</td>
<td>37.98</td>
<td>220.6</td>
<td>101</td>
<td>18.75</td>
<td>203.9</td>
<td>118</td>
<td>9.18</td>
<td>6.3</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>MA</td>
<td>121 354-8285</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>0</td>
<td>218.2</td>
<td>88</td>
<td>37.09</td>
<td>348.5</td>
<td>108</td>
<td>29.62</td>
<td>212.6</td>
<td>118</td>
<td>9.57</td>
<td>7.5</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>NO</td>
<td>147 333-0021</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>0</td>
<td>157</td>
<td>70</td>
<td>25.69</td>
<td>103.1</td>
<td>94</td>
<td>8.76</td>
<td>211.8</td>
<td>96</td>
<td>0.53</td>
<td>7.1</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>LA</td>
<td>117 335-4719</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>0</td>
<td>154.5</td>
<td>97</td>
<td>31.37</td>
<td>35.18</td>
<td>60</td>
<td>29.69</td>
<td>215.8</td>
<td>90</td>
<td>9.71</td>
<td>8.7</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>WA</td>
<td>141 382-8783</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>0</td>
<td>258.2</td>
<td>71</td>
<td>43.68</td>
<td>332.0</td>
<td>111</td>
<td>18.87</td>
<td>326.4</td>
<td>67</td>
<td>4.69</td>
<td>11.2</td>
<td>6</td>
</tr>
</tbody>
</table>
Data After Preparation for Spark using Python

Notes:
The columns with data issues have been removed—State, Phone, zip and all charges
The target variable has been converted to 0,1 and must be the last column
The other two flag columns (IntlPlan and Vmail Plan) have been set to 0,1 as well
Headings removed
Steps to Accomplish

1. Create Notebook in IBM Data Science
2. Load Data
3. Insert code to read data
4. Parse the data
5. Run decision tree
Master the art of data science

Solve your toughest data challenges with the best tools and the latest expertise in a social environment built by data scientists.

Watch Now (2:31)

Sign Up For a Free Trial
IBM DSX – Create New Project/Notebook
Create Notebook

Blank  From File  From URL

Name*
Decision Tree

37 Characters Remaining

Description

Type your Description here

Language*
- Python 2
- R
- Scala
- Python 3.5 Experimental
Select Project & Spark Service

Create Notebook

Create Notebook

**Blank**  **From File**  **From URL**

**Language**
- Python 2
- R
- Scala
- Python 3.5 Experimental

**Spark version**
- 2.1
- 2.0
- 1.6

**Project**
- Churn

Add the notebook to an existing project.

**Spark Service**
- Apache Spark-2.0

Associate this notebook with the Spark Service of your choice.

[Create Notebook]
Upload a Data File
Insert Data to Code

Drop your file here or browse your files to add a new file.

- churn-decision-tree.csv

  Insert to code

  - Insert Pandas DataFrame
  - Insert Spark SQL DataFrame
  - Insert Spark RDD
  - Insert Credentials
from pyspark.mllib.regression import LabeledPoint
import random
import requests, StringIO, pandas as pd, json, re

def set_hadoop_config(creds):
    """This function sets the Hadoop configuration with given credentials, 
    so it is possible to access data using SparkContext""

    prefix = "fs.swift.service." + creds['name']
    hconf = sc._jsc.hadoopConfiguration()
    hconf.set(prefix + ".auth.url", creds['auth_url']+'/v3/auth/tokens')
    hconf.set(prefix + ".auth.endpoint.prefix", "endpoints")
    hconf.set(prefix + ".tenant", creds['project_id'])
    hconf.set(prefix + ".username", creds['user_id'])
    hconf.set(prefix + ".password", creds['password'])
    hconf.setInt(prefix + ".http.port", 8080)
    hconf.set(prefix + ".region", creds['region'])
    hconf.setBoolean(prefix + ".public", True)

# @hidden_cell
credentials_1 = {
    'auth_url': 'https://identity.open.softlayer.com',
    'project': 'object_storage_737bb791_2163_44dd_954b_5c6008e765d7',
    'project_id': 'a3b9e9e4df854805b0b1e69fdf207588',
    'region': 'dallas',
    'user_id': 'e93b44f6beb643b3bab95720c82dbb75',
    'domain_id': 'cb31f05ee2a540459797d4b2b22b189a',
    'domain_name': '1149847',
    'username': 'member_f338ba17e0276ed7040d0a8541fb0a8ff8d605fa',
    'password': '1Ch7j8AX4=N8#3',
    'container': 'Analytics',
    'tenantId': 'undefined',
    'filename': 'churn-decision-tree.csv'
}
credentials_1['name'] = 'keystone'
set_hadoop_config(creds_1)
# Load and parse the data

def parsePoint(line):
    values = [float(x) for x in line.split(",")]
    return LabeledPoint(values[13], values[0:12])

data = sc.textFile("swift://" + credentials_1['container'] + "." + credentials_1['name'] + "/" + credentials_1['filename'])
parsedData = data.map(parsePoint)
Code for Building Decision Tree

```python
#Split data approximately into training (70%) and test (30%)
training, test = parsedData.randomSplit([0.7, 0.3], seed=0)

#Decision tree classification
from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
from pyspark.mllib.util import MLUtils

#Train a decision tree model
#Empty categoricalFeaturesInfo indicates all features are continuous
model = DecisionTree.trainClassifier(training, numClasses=2, categoricalFeaturesInfo={}, impurity='gini', maxDepth=5, maxBins=32)

#Evaluate model on test instances and compute test error
predictions = model.predict(test.map(lambda x: x.features))
labelsAndPredictions = test.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(test.count())

print('Test Error = ' + str(testErr))
print('Learned classification tree model: ')
print(model.toDebugString())
```
Decision Tree Results

Test Error = 0.0837438423645

Learned classification tree model:
DecisionTreeModel classifier of depth 5 with 45 nodes

If (feature 4 <= 264.7)
  If (feature 1 <= 0.0)
    If (feature 4 <= 242.2)
      If (feature 4 <= 138.5)
        If (feature 11 <= 3.0)
          Predict: 0.0
        Else (feature 11 > 3.0)
          Predict: 0.0
      Else (feature 4 > 138.5)
        If (feature 6 <= 267.3)
          Predict: 0.0
        Else (feature 6 > 267.3)
          Predict: 0.0
    Else (feature 4 > 242.2)
      If (feature 6 <= 240.9)
        If (feature 8 <= 241.3)
          Predict: 0.0
        Else (feature 8 > 241.3)
          Predict: 0.0
      Else (feature 6 > 240.9)
        If (feature 2 <= 0.0)
          Predict: 1.0
        Else (feature 2 > 0.0)
          Predict: 0.0
  Else (feature 1 > 0.0)
    If (feature 11 <= 2.0)
      Predict: 1.0
    Else (feature 11 > 2.0)
      If (feature 10 <= 13.0)
        If (feature 8 <= 181.1)
          Predict: 0.0
        Else (feature 8 > 181.1)
          Predict: 0.0
    Else (feature 11 > 2.0)
Put Predicted and Actual Values in Arrays

# converts to array
predictions_array = predictions.collect()

# extracts actual Churn values from test data
actual_test = test.map(lambda lp: lp.label)
actual_array = actual_test.collect()
Code for ROC Chart

from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
import random

false_positive_rate, true_positive_rate, threshold = roc_curve(actual_array, predictions_array)

print(false_positive_rate)
print(true_positive_rate)
roc_auc = auc(false_positive_rate, true_positive_rate)
plt.title('Receiver Operating Characteristic for test data')
plt.plot(false_positive_rate, true_positive_rate, 'b', label='AUC = %0.2f' % roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1], 'r--')
plt.xlim([-0.1,1.0])
plt.ylim([-0.1,1.0])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
The ROC Chart

Training Data

Test (or validation) Data

Receiver Operating Characteristic for test data

Receiver Operating Characteristic training data

AUC = 0.72

AUC = 0.76
Neural Network

1. Create Notebook in IBM Data Science Experience
2. Load Data
3. Insert Code to Read Data
4. Prepare the Data
5. Run Neural Network
Code to Read Data

```python
from io import StringIO
import requests
import json
import pandas as pd

def get_object_storage_file_with_credentials_737bb79121634dd954b5c6008e765d7(container, filename):
    
    # This function returns a StringIO object containing
    # the file content from Bluemix Object Storage.

    url1 = ''.join(['https://identity.open.softlayer.com', '/v3/auth/tokens'])
    data = {'auth': {'identity': {'methods': ['password']},
                     'password': {'user': {'name': 'member_f338ba17e0276ed7040d0a8541fb0a8ff8d05fa', 'domain': {'id': 'cb31f05ee2a540450797d4b1b2b2189a'},
                                      'password': '1Ch7jRAX4N8#J'}}})
    headers1 = {'Content-Type': 'application/json'}
    resp1 = requests.post(url=url1, data=json.dumps(data), headers=headers1)
    resp1_body = resp1.json()
    for e1 in resp1_body['token']['catalog']:
        if e1['type']=='object-store':
            for e2 in e1['endpoints']:
                if e2['interface']=='public' and e2['region']=='dallas':
                    url2 = ''.join([e2['url'], '/', container, '/', filename])
                    s_subject_token = resp1.headers['x-subject-token']
                    headers2 = {'X-Auth-Token': s_subject_token, 'accept': 'application/json'}
                    resp2 = requests.get(url=url2, headers=headers2)
                    return StringIO(resp2.text)

df_data_3 = pd.read_csv(get_object_storage_file_with_credentials_737bb79121634dd954b5c6008e765d7('Analytics', 'Churn_with_headers.csv'))
```
Prepare the Data (Standardization)

```python
from sklearn import preprocessing

std_scale = preprocessing.StandardScaler().fit(df_data_3[['Account_length']])
df_data_3['Account_length'] = std_scale.transform(df_data_3[['Account_length']])
```

Result -

<table>
<thead>
<tr>
<th>Account_length</th>
<th>Int1Plan</th>
<th>Vmail Plan</th>
<th>Vmail Message</th>
<th>Day Mins</th>
<th>day Calls</th>
<th>Eve Mins</th>
<th>Eve Calls</th>
<th>Night Mins</th>
<th>Night Calls</th>
<th>Int1 Mins</th>
<th>Int1 Calls</th>
<th>CustServ Calls</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.676489</td>
<td>0</td>
<td>0</td>
<td>1.234883</td>
<td>1.555767</td>
<td>0.475643</td>
<td>-0.070610</td>
<td>-0.055940</td>
<td>0.866743</td>
<td>-0.465494</td>
<td>-0.085008</td>
<td>-0.601195</td>
<td>-0.427932</td>
<td>0</td>
</tr>
<tr>
<td>0.149065</td>
<td>0</td>
<td>1</td>
<td>1.307948</td>
<td>-0.333736</td>
<td>1.124503</td>
<td>-0.108080</td>
<td>0.144867</td>
<td>1.058571</td>
<td>0.147825</td>
<td>1.240482</td>
<td>-0.601195</td>
<td>-0.427932</td>
<td>0</td>
</tr>
<tr>
<td>0.902529</td>
<td>0</td>
<td>0</td>
<td>-0.591760</td>
<td>1.168304</td>
<td>0.675985</td>
<td>-1.573383</td>
<td>0.492679</td>
<td>-0.766869</td>
<td>0.198935</td>
<td>0.703121</td>
<td>0.211534</td>
<td>-1.188218</td>
<td>0</td>
</tr>
<tr>
<td>-0.428690</td>
<td>1</td>
<td>0</td>
<td>-0.591760</td>
<td>2.195596</td>
<td>-1.466936</td>
<td>-2.742665</td>
<td>-0.608159</td>
<td>-0.078551</td>
<td>-0.567714</td>
<td>-1.303026</td>
<td>1.024263</td>
<td>0.332354</td>
<td>0</td>
</tr>
<tr>
<td>-0.654629</td>
<td>1</td>
<td>0</td>
<td>-0.591760</td>
<td>-0.240900</td>
<td>0.525149</td>
<td>-1.038932</td>
<td>1.098699</td>
<td>-0.276311</td>
<td>1.067803</td>
<td>-0.049184</td>
<td>-0.601195</td>
<td>1.092641</td>
<td>0</td>
</tr>
<tr>
<td>0.425335</td>
<td>1</td>
<td>0</td>
<td>-0.591760</td>
<td>0.801056</td>
<td>-0.121381</td>
<td>0.386928</td>
<td>0.044463</td>
<td>0.059881</td>
<td>0.914473</td>
<td>-1.410498</td>
<td>0.617898</td>
<td>-1.188218</td>
<td>0</td>
</tr>
<tr>
<td>0.500681</td>
<td>0</td>
<td>1</td>
<td>1.161817</td>
<td>0.705572</td>
<td>-0.619736</td>
<td>2.909300</td>
<td>0.395875</td>
<td>0.231933</td>
<td>0.914473</td>
<td>-0.980610</td>
<td>1.024263</td>
<td>1.092641</td>
<td>0</td>
</tr>
<tr>
<td>1.153683</td>
<td>1</td>
<td>0</td>
<td>-0.591760</td>
<td>-0.418205</td>
<td>-1.068253</td>
<td>-1.930341</td>
<td>-0.306949</td>
<td>0.216112</td>
<td>-0.205945</td>
<td>-1.123906</td>
<td>0.617898</td>
<td>-1.188218</td>
<td>0</td>
</tr>
</tbody>
</table>
Prepare the Data (Combining Predictors to Features)

1. Convert to Pandas Data Frame

```python
pDF = sqlContext.createDataFrame(df_data_3)
pDF.show()
```

2. Combine the predictors to features column

```python
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(
    inputCols=["Account_length", "IntlPlan", "Vmail Plan", "Vmail Message", "Day Mins", "day Calls","Eve Mins","Eve Calls","Night Mins","Night Calls"],
    outputCol="features")

output = assembler.transform(pDF)
print("Assembled columns to vector column 'features'")
#output.select("features").show(truncate=False)
output.show()
```
Run Neural Network

```python
from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Split the data into train and test
splits = output.randomSplit([0.6, 0.4], 1234)
train = splits[0]
test = splits[1]

# specify layers for the neural network:
# input layer of size 4 (features), two intermediate of size 5 and 4
# and output of size 2 (classes)
layers = [13, 3, 2]

# create the trainer and set its parameters
trainer = MultilayerPerceptronClassifier(maxIter=100, layers=layers, blockSize=128, seed=1234)

# train the model
model = trainer.fit(train)

# compute accuracy on the test set
result = model.transform(test)
predictionAndLabels = result.select("prediction", "label")
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
print("Test set accuracy = " + str(evaluator.evaluate(predictionAndLabels)))
```

Result - Test set accuracy = 0.920792079208
Dillard’s Returns Data Set

<table>
<thead>
<tr>
<th>Tran_Type</th>
<th>TranAmt</th>
<th>Online</th>
<th>Tender_Ty</th>
<th>Deptcent</th>
<th>Distance_</th>
<th>Division</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>0.01</td>
<td>N</td>
<td>BANK</td>
<td>CHILDREN</td>
<td>6.381</td>
<td>4</td>
<td>TN</td>
</tr>
<tr>
<td>P</td>
<td>0.01</td>
<td>N</td>
<td>BANK</td>
<td>CHILDREN</td>
<td>2.053</td>
<td>7</td>
<td>TX</td>
</tr>
<tr>
<td>R</td>
<td>0.01</td>
<td>N</td>
<td>BANK</td>
<td>COSMETIC</td>
<td>3.009</td>
<td>9</td>
<td>ID</td>
</tr>
<tr>
<td>P</td>
<td>0.01</td>
<td>N</td>
<td>BANK</td>
<td>CHILDREN</td>
<td>2.797</td>
<td>9</td>
<td>AZ</td>
</tr>
<tr>
<td>R</td>
<td>0.01</td>
<td>N</td>
<td>BANK</td>
<td>COSMETIC</td>
<td>4.165</td>
<td>9</td>
<td>NV</td>
</tr>
<tr>
<td>P</td>
<td>0.01</td>
<td>N</td>
<td>BANK</td>
<td>CHILDREN</td>
<td>5.821</td>
<td>4</td>
<td>KY</td>
</tr>
<tr>
<td>R</td>
<td>0.01</td>
<td>N</td>
<td>BANK</td>
<td>COSMETIC</td>
<td>4.871</td>
<td>4</td>
<td>MO</td>
</tr>
<tr>
<td>P</td>
<td>0.01</td>
<td>N</td>
<td>BANK</td>
<td>CHILDREN</td>
<td>7.072</td>
<td>4</td>
<td>AR</td>
</tr>
<tr>
<td>R</td>
<td>0.01</td>
<td>N</td>
<td>BANK</td>
<td>READY-TO</td>
<td>2.678</td>
<td>4</td>
<td>AR</td>
</tr>
<tr>
<td>P</td>
<td>0.01</td>
<td>N</td>
<td>BANK</td>
<td>CHILDREN</td>
<td>18.5</td>
<td>4</td>
<td>KS</td>
</tr>
<tr>
<td>R</td>
<td>0.01</td>
<td>N</td>
<td>DAMX</td>
<td>LING/ACC</td>
<td>12.471</td>
<td>9</td>
<td>AZ</td>
</tr>
<tr>
<td>P</td>
<td>0.01</td>
<td>N</td>
<td>BANK</td>
<td>CHILDREN</td>
<td>31.587</td>
<td>4</td>
<td>AR</td>
</tr>
<tr>
<td>R</td>
<td>0.01</td>
<td>N</td>
<td>DLRD</td>
<td>DECOR. HO</td>
<td>22.777</td>
<td>7</td>
<td>OK</td>
</tr>
</tbody>
</table>

- 8 Columns
- 1 million rows
- Target Variable: Tran_Type – R/P
from StringIO import StringIO
import requests
import json
import pandas as pd

def get_object_storage_file_with_credentials_aad8c9dfe51c428599cdcc548df1f5bc(container, filename):
    """This function returns a StringIO object containing
    the file content from Bluemix Object Storage."""

    url1 = ''.join(['https://identity.open.softlayer.com', '/v3/auth/tokens'])
    data = {'auth': {'identity': {'methods': ['password'],
                                'password': {'user': {'name': 'member_e16ba04fa120640b5804d46cc61b2ae2d9532fb', 'domain': {'id': '6dfe9653228d4ed2933dac8c22f464a7'},
                                'password': 'ZA_5m4P6(GszrD[K])'}}}}
    headers1 = {'Content-Type': 'application/json'}
    resp1 = requests.post(url=url1, data=json.dumps(data), headers=headers1)
    resp1_body = resp1.json()

    for e1 in resp1_body['token']['catalog']:
        if e1['type'] == 'object-store':
            for e2 in e1['endpoints']:
                if e2['interface'] == 'public' and e2['region'] == 'dallas':
                    url2 = ''.join([e2['url'], '/', container, '/', filename])
                    s_subject_token = resp1.headers['x-subject-token']
                    headers2 = {'X-Auth-Token': s_subject_token, 'accept': 'application/json'}
                    resp2 = requests.get(url=url2, headers=headers2)
                    s_content = resp2.text

    return StringIO(s_content)

data_1 = get_object_storage_file_with_credentials_aad8c9dfe51c428599cdcc548df1f5bc('DEDDillardsData1', 'Dillards_returns_C.txt')
df_data = pd.read_csv(data_1)
Prepare the Data

Convert String to Numeric

def tran_type_to_numeric(x):
    if x=='R':
        return 0
    if x=='P':
        return 1

def tender_type_to_numeric(x):
    if x=='BANK':
        return 0
    if x=='DLRD':
        return 1
    if x=='DAMX':
        return 2
    if x=='CASH':
        return 3
    else:
        return 4

df_data['Tender_Type'] = df_data['Tender_Type'].apply(tender_type_to_numeric)
df_data['Tran_Type'] = df_data['Tran_Type'].apply(tran_type_to_numeric)
# After Conversion

<table>
<thead>
<tr>
<th>Tran_Type</th>
<th>TranAmt</th>
<th>Online</th>
<th>Tender_Type</th>
<th>Deptcnet_Desc</th>
<th>Distance_to_Nearest_Store</th>
<th>Division</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6.381</td>
<td>4</td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.053</td>
<td>7</td>
<td>26</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3.009</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2.797</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4.165</td>
<td>9</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5.821</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4.871</td>
<td>4</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>7.072</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8.500</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>12.471</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>18.500</td>
<td>4</td>
<td>11</td>
</tr>
</tbody>
</table>
Run the Decision Tree

```
from pyspark.mllib.regression import LabeledPoint
pDF = sqlContext.createDataFrame(df_data)
temp = pDF.rdd.map(lambda line: LabeledPoint(line[0],[line[1:]]))

#Split data approximately into training (60%) and test (40%)
training, test = temp.randomSplit([0.6, 0.4], seed=0)

#Decision tree classification
from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
from pyspark.mllib.util import MLUtils

#Train a decision tree model
#Empty categoricalFeaturesInfo indicates all features are continuous
model = DecisionTree.trainClassifier(training, numClasses=2, categoricalFeaturesInfo={}, impurity='gini', maxDepth=5, maxBins=32)

#Evaluate model on test instances and compute test error
predictions = model.predict(test.map(lambda x: x.features))
labelsAndPredictions = test.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(test.count())

print('Test Error = ' + str(testErr))
print('Learned classification tree model:

print(model.toDebugString())
```

Test Set Accuracy = 0.981284
Neural Network – Code to Read Data

```python
from io import StringIO
import requests
import json
import pandas as pd

def get_object_storage_file_with_credentials_aad8c9dfe51c428599cdcc548df1f5bc(container, filename):
    """This function returns a StringIO object containing
    the file content from Bluemix Object Storage."""

    url1 = 'https://identity.open.softlayer.com/' + '/v3/auth/tokens
    data = {'auth': {'identity': {'methods': ['password']},
                     'password': {'user': {'name': 'member_e16ba8e4fa120640b5804d45cc61b2ae2d9532fb', 'domain': {'id': '6dfe9653228d4ed29333dac822f464a7'}},
                                     'password': 'ZA_5r4P[GszwD_[K]]')}
    headers1 = {'Content-Type': 'application/json'}
    resp1 = requests.post(url=url1, data=json.dumps(data), headers=headers1)
    resp1_body = resp1.json()

    for e1 in resp1_body['token']['catalog']:
        if e1['type'] == 'object-store':
            for e2 in e1['endpoints']:
                if e2['interface'] == 'public' and e2['region'] == 'dallas':
                    url2 = '/'.join([e2['url'], '/', container, '/', filename])
                    s_subject_token = resp1.headers['x-subject-token']
                    headers2 = {'X-Auth-Token': s_subject_token, 'accept': 'application/json'}
                    resp2 = requests.get(url=url2, headers=headers2)
                    return StringIO(resp2.text)

data_1 = get_object_storage_file_with_credentials_aad8c9dfe51c428599cdcc548df1f5bc('DEDDillardsData1', 'Dillards_returns_C.txt')

df_data = pd.read_csv(data_1)
```
Prepare the Data

Convert String to Numeric

def tran_type_to_numeric(x):
    if x=='R':
        return 0
    if x=='P':
        return 1

def tender_type_to_numeric(x):
    if x=='BANK':
        return 0
    if x=='DLRD':
        return 1
    if x=='DAMX':
        return 2
    if x=='CASH':
        return 3
    else:
        return 4

df_data['Tender_Type'] = df_data['Tender_Type'].apply(tender_type_to_numeric)
df_data['Tran_Type'] = df_data['Tran_Type'].apply(tran_type_to_numeric)
After Conversion

<table>
<thead>
<tr>
<th>Tran_Type</th>
<th>TranAmt</th>
<th>Online</th>
<th>Tender_Type</th>
<th>Deptcent_Desc</th>
<th>Distance_to_Nearest_Store</th>
<th>Division</th>
<th>State</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6.381</td>
<td>4</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.053</td>
<td>7</td>
<td>26</td>
<td>1</td>
</tr>
<tr>
<td>R</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3.009</td>
<td>9</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.797</td>
<td>9</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>R</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4.165</td>
<td>9</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5.821</td>
<td>4</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>R</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4.871</td>
<td>4</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7.072</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>R</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.678</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18.500</td>
<td>4</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>R</td>
<td>0.01</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>12.471</td>
<td>9</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Note – Here we have a new column called “label”. It contains the target variables. Neural Network accepts two columns – label(Target) and features(Predictors)
Combining Predictors to Features

```python
pDF = sqlContext.createDataFrame(df_data)

from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(
    inputCols=["TranAmt", "Online", "Tender_Type", "Deptcent_Desc", "Distance_to_Nearest_Store", "Division","State"],
    outputCol="features")

output = assembler.transform(pDF)
print("Assembled columns to vector column 'features'")
#output.select("features").show(truncate=False)
output.show()
```
Run the Neural Network

```python
from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Split the data into train and test
splits = output.randomSplit([0.6, 0.4], 1234)
train = splits[0]
test = splits[1]

# specify layers for the neural network:
# input layer of size 4 (features), two intermediate of size 5 and 4
# and output of size 2 (classes)
layers = [13, 3, 2]

# create the trainer and set its parameters
trainer = MultilayerPerceptronClassifier(maxIter=100, layers=layers, blockSize=128, seed=1234)

# train the model
model = trainer.fit(train)

# compute accuracy on the test set
result = model.transform(test)
predictionAndLabels = result.select("prediction", "label")
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
print("Test set accuracy = " + str(evaluator.evaluate(predictionAndLabels)))

Test set accuracy = 0.98447840272
```
Decision Tree and Neural Network never reached completion with 13 million records
# Summary

<table>
<thead>
<tr>
<th></th>
<th>SPSS Modeler</th>
<th>IBM Spark/Python</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (secs)</td>
<td>Accuracy(%)</td>
</tr>
<tr>
<td></td>
<td>(AUC)</td>
<td></td>
</tr>
<tr>
<td>Churn Data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DT</td>
<td>01.89 (.876)</td>
<td>93.79</td>
</tr>
<tr>
<td>NN</td>
<td>02.08 (.889)</td>
<td>93.80</td>
</tr>
<tr>
<td>Dillard’s Data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DT</td>
<td>14.28 (.989)</td>
<td>96.85</td>
</tr>
<tr>
<td>NN</td>
<td>448.00 (.990)</td>
<td>98.70</td>
</tr>
</tbody>
</table>
Conclusion

• Regardless of the tool used to build the models (Modeler, R, Python, ...), all the critical concepts still apply—domain knowledge, data exploration, dimension reduction, preventing overfit models, assessment of models
Conclusion (cont)

Both are Important

• IBM SPSS Modeler
  – Data Analyst
  Faster model development, trusted results, useful output, extensive data sources including Hadoop, easier data exploration and standardization as well as scoring
  – Data Science
  Except for special data extraction and those who think they must code everything

• IBM Data Science
  (Note this is a developer environment)
  – Specialized data extraction
  – Newer algorithms not yet generally incorporated in SPSS Modeler
  – Coding required to incorporate into application
  – Those that believe they need to code everything (really because they just like to code)
Gartner Technology Hype Cycle