THE POWER TO KNOW®

Machine Learning Using SAS® Viya®

Course Notes

Machine Learning Using SAS® Viya® Course Notes was developed by Carlos Pinheiro, Andy Ravenna, Sharad Saxena, and Jeff Thompson. Additional contributions were made by Marya Ilgen-Lieth and Cat Truxillo. Instructional design, editing, and production support was provided by the Learning Design and Development team.

Machine Learning Using SAS® Viya® Course Notes

Book code E71496, course code LWCPML3V/CPML3V, prepared date 23Apr2019. LWCPML3V_001

ISBN 978-1-64295-361-9

Table of Contents

To learn more…

For information about other courses in the curriculum, contact the SAS Education Division at 1-800-333-7660, or send e-mail to training@sas.com. You can also find this information on the web at http://support.sas.com/training/ as well as in the Training Course Catalog.

For a list of SAS books (including e-books) that relate to the topics covered in this course notes, visit https://www.sas.com/sas/books.html or call 1-800-727-0025. US customers receive free shipping to US addresses.

Chapter 1 Introduction

1.1 Machine Learning in Business Decision Making

> “In the new world, it is not the big fish which eats the small fish, it’s the fast fish which eats the slow fish.”

Klaus Schwab
Founder and Executive Chairman
World Economic Forum

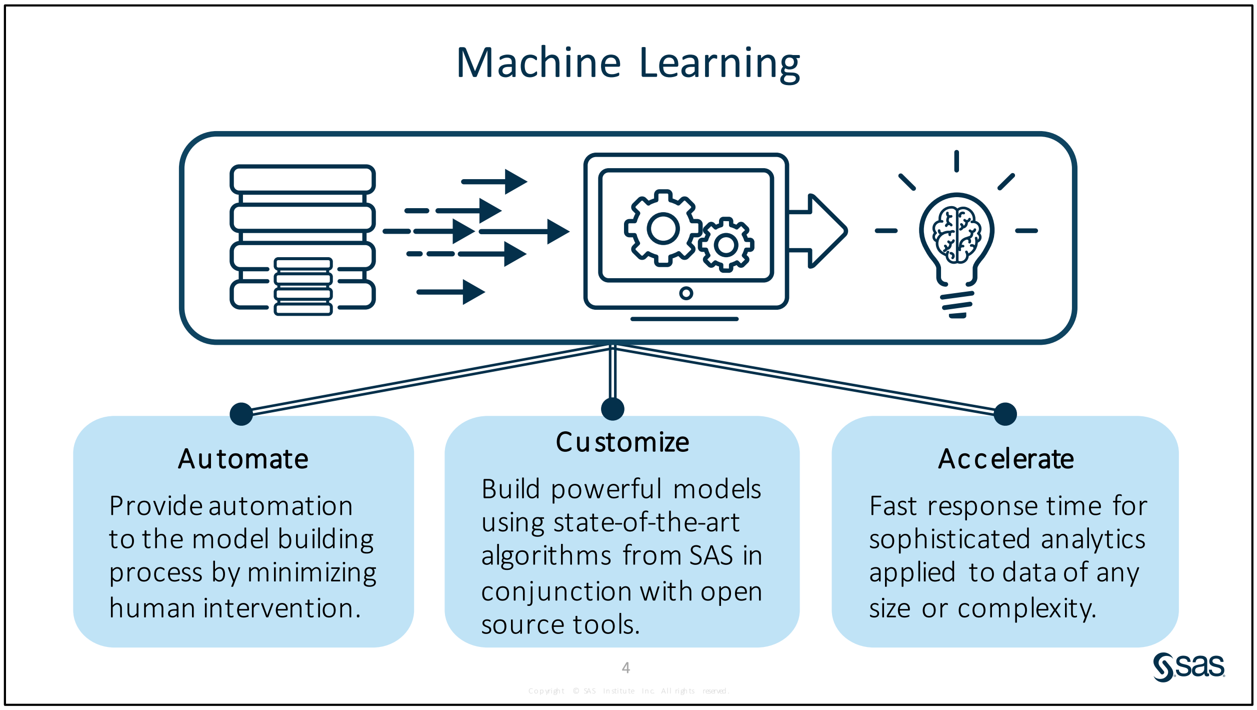
3

Copyright © SAS Institute Inc. All rights reserved.

In today’s marketplace, all you need is to deliver fast results with a lot of data!

The message here is about the importance of speed to deliver the models in production, considering the entire process of identifying the information needed to proceed the model: preparing the data; training, validating, and testing the model; and finally deploying it in production.

Organizations in very dynamic and competitive markets require even more velocity in such an analytical cycle, and machine learning is the center of focus to achieve this.



Machine Learning

Automate

Provide automation to the model building process by minimizing human intervention.

Customize

Build powerful models using state-of-the-art algorithms from SAS in conjunction with open source tools.

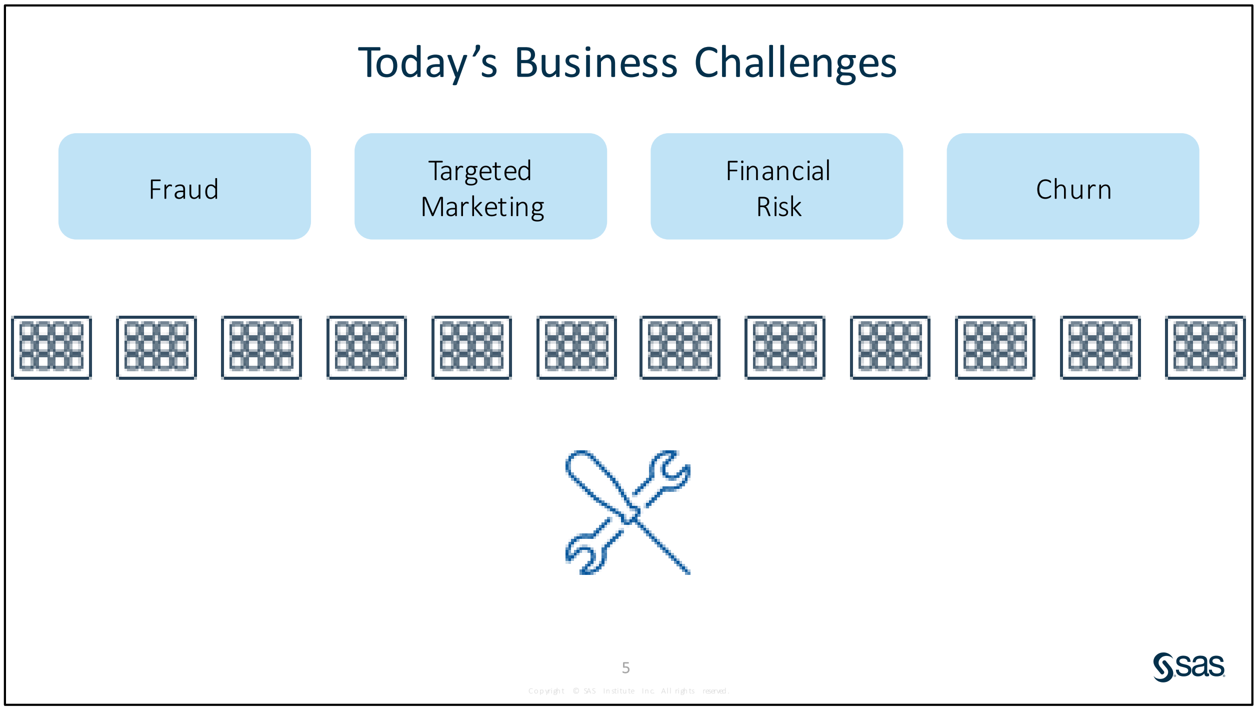
Accelerate

Fast response time for sophisticated analytics applied to data of any size or complexity.

4

Machine learning is a branch of artificial intelligence that automates the building of systems that learn iteratively from data, identify patterns, and predict future results – with minimal human intervention. It shares many approaches with other related fields, but it focuses on predictive accuracy rather than interpretability of the model.

Building representative machine learning models that generalize well on future data requires careful consideration of both the data at hand and assumptions about the various available training algorithms.



Today’s Business Challenges

Fraud

Targeted Marketing

Financial Risk

Churn

5

Listed here are some use cases in the context of machine learning. Churn, fraud, targeted marketing, and financial risk are common application areas.

Fraud detection methods attempt to detect or impede illegal activity involving financial transactions. Fraud is intentionally committing a misleading act for financial gain. Anomaly detection is one of the ways to detect fraud. You look to predict an event that occurs rarely and identify patterns in the data that do not conform to expected behavior, such as abnormally high purchase made on a credit card. This is where machine learning algorithms can be used effectively.

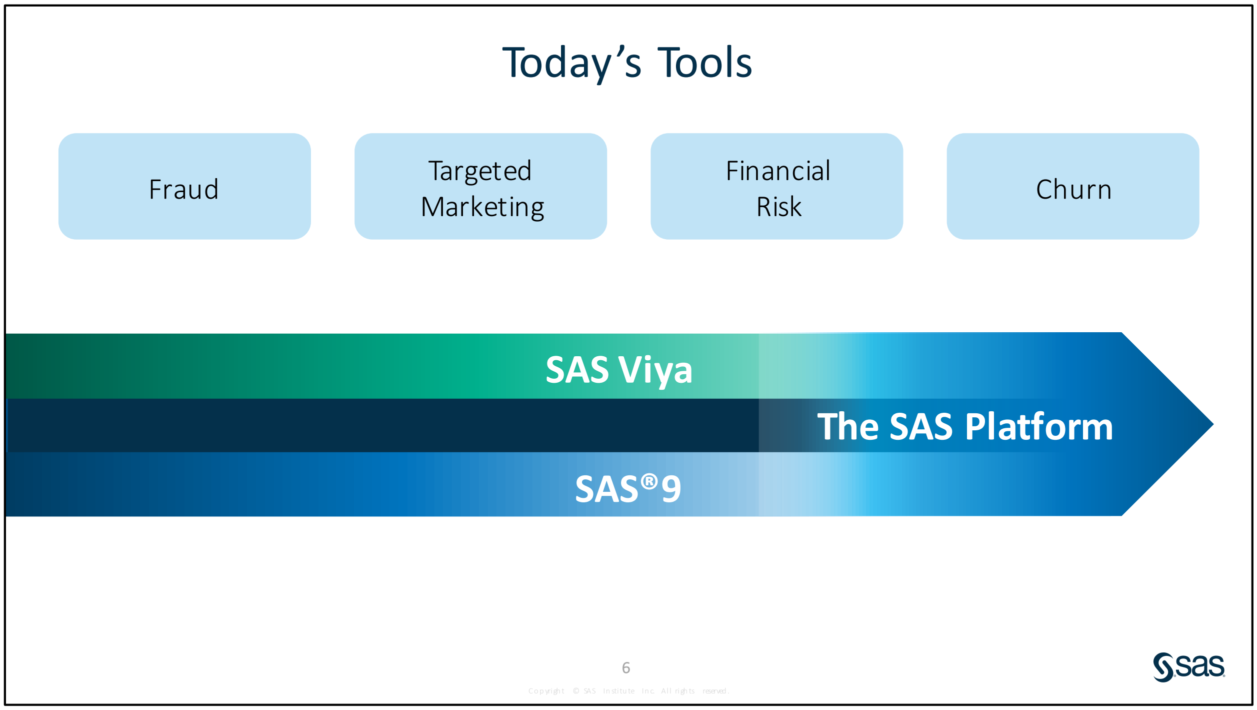
Targeted marketing is one more common application area. Most companies rely on some form of direct marketing to acquire new customers and generate additional revenue from existing customers. Predictive modeling generally accomplishes this by helping companies answer crucial questions such as the following: Who should I contact? What should I offer? When should I make the offer? How should I make the offer?

Financial risk management models attempt to predict monetary events such as credit default, loan prepayment, and insurance claim. Banks use multiple models to meet a variety of regulations (such as CCAR and Basel III). With increased scrutiny on model risk, bankers must establish a model risk management program for regulatory compliance and business benefits. Models are useful things to have around and bankers have come to rely on them for certain applications, some of which expose the bank to significant risks. Predictive models fall into this category. Examples include loan approval using credit scoring and hedging models using swaps and options to manage the balance sheet while protecting liquidity and determining capital adequacy.

Churn or attrition is the turnover of customers of a product or users of a service. Churn is the focus in this course and is discussed soon.

This list is not exhaustive. There are many more applications that machine learning impacts today. For example, database marketing applications include offer response, up-sell, cross-sell, and attrition models. Process monitoring applications detect deviations from the norm in manufacturing, financial, and security processes. Pattern detection models are used in applications ranging from handwriting analysis to medical diagnostics.

Recommendations, text mining, and predictive asset maintenance are some others, to name a few.



Today’s Tools

Fraud

Targeted Marketing

Financial Risk

Churn

SAS Viya

The SAS Platform

SAS®9

6

Copyright © SAS Institute Inc. All rights reserved.

The SAS Platform is a software foundation that is engineered to address today's business challenges and to generate insights from your data in any computing environment. SAS Viya is the latest extension of the SAS Platform, which is designed to enable analytics to the enterprise. SAS Viya seamlessly scales for data of any size, type, speed, and complexity and is interoperable with SAS®9.

Here are some advantages of the SAS Platform:

Diversity

- Makes analytics accessible to anyone seeking insights from data – regardless of skill or experience – via a rich interface.
- Easily tackles both modern and legacy data types so that you can unify the data that you need to generate successful actions from analytics.
- Handles analytical problems of any size or complexity with an extensive suite of proven methods.
- Executes in a cloud, on-site, or hybrid environment. Deploys seamlessly to any infrastructure or application ecosystem.
- Adapts to the full spectrum of analytics and data challenges that you face, and embraces open source technology with interfaces to consistent, governed code.

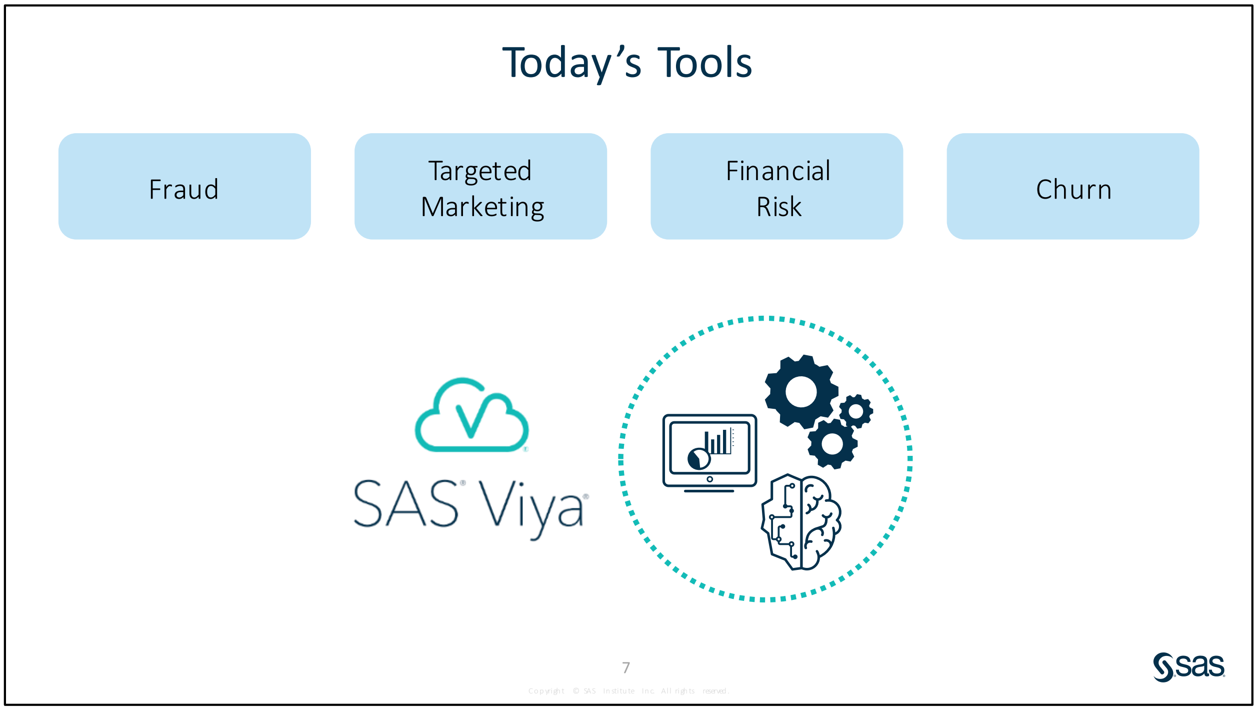
Scale

- Scales from small to exceedingly big data, in motion and at rest, at the edge and in the cloud.
- Provides processing power optimized for analytic workloads that is easily transferable and right-sized for both physical and virtual hardware environments.
- Industrializes analytics for thousands of concurrent models and simultaneous users. Gives you fast, traceable results with centralized monitoring and governance.
- Ensures continual insights with IT-controlled, enterprise-class analytics fortified with failover protection, infrastructure pooling, and high availability.
- Delivers answers from complex analytics fast, taking advantage of parallel, in-memory and distributed computing environments, and a common, governed code base.

Trust

- Integrates the entire analytics life cycle – from data, to discovery, to deployment. Delivers a traceable results path from start to finish, visible to any audit.
- Ensures resilience to infrastructure changes with portable code, along with built-in intelligent defaults and automated analytics tasks.
- Builds on the experience and expertise of more than four decades of deployments in virtually every industry, applying analytics to myriad business issues.
- Verifies the accuracy and validity of analytical results with a rigorous testing framework and proven domain practices.
- Includes security features to help guard against potential areas of vulnerability with options for passwords, data, and users.

You can achieve excellence in machine learning analytics with the SAS Platform that supports diversity, enables scale, and promotes trust.



Today’s Tools

Fraud

Targeted Marketing

Financial Risk

Churn

SAS® Viya®

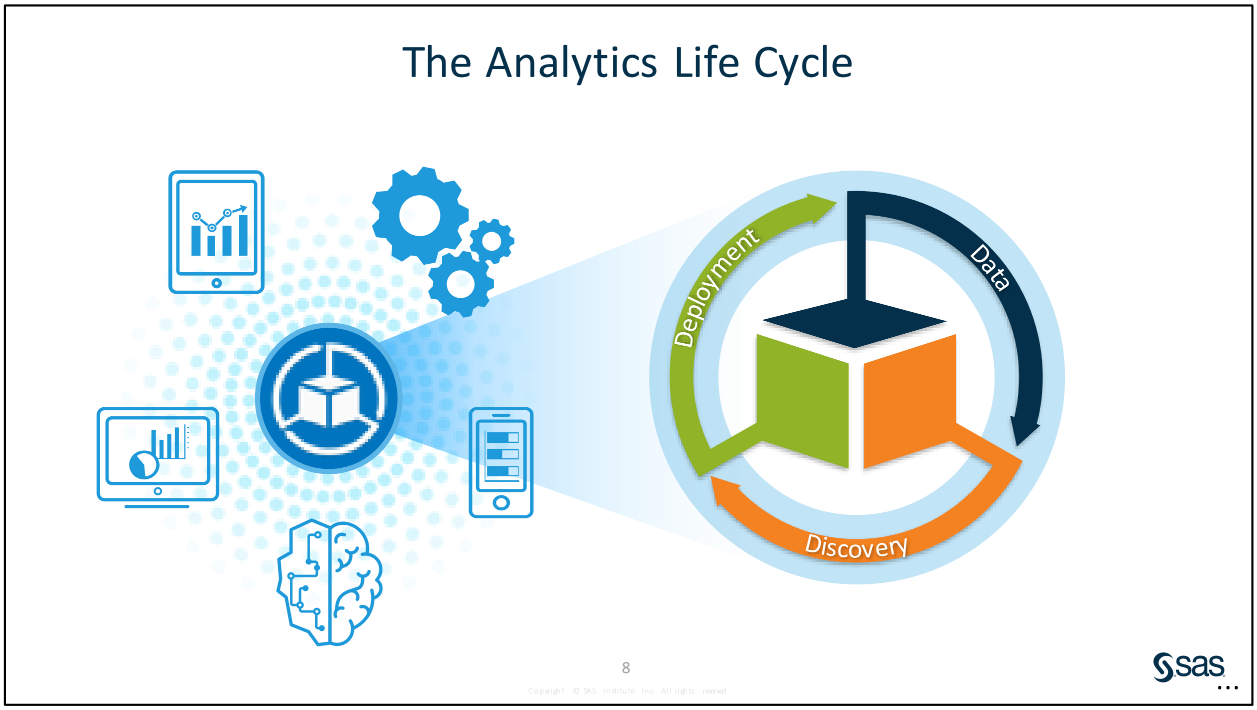
7

sas

In this course, you learn to use SAS Viya to create supervised machine-learning models. You need to go from data to decisions as quickly as possible. Machine learning models are at the heart of critical business decisions. They can identify new opportunities and enable you to manage uncertainty and risks. To create these models, you need powerful and easy-to-use software that can help you wrangle your data into shape and quickly create many accurate predictive models. You also need an integrated process to manage your analytical models for optimal performance throughout their lifespan. SAS Viya provides efficient, repeatable processes and a reliable architecture for managing data, communicating the rationale, and tracing the predictive analytics models through the deployment phase.

SAS Viya is a cloud-enabled, in-memory analytics engine that delivers everything you need for quick, accurate, and consistent results. Elastic, scalable, and fault-tolerant processing addresses the complex analytical challenges of today and effortlessly scales to meet your future needs.

Note: SAS Viya is covered in detail in this chapter.



The Analytics Life Cycle

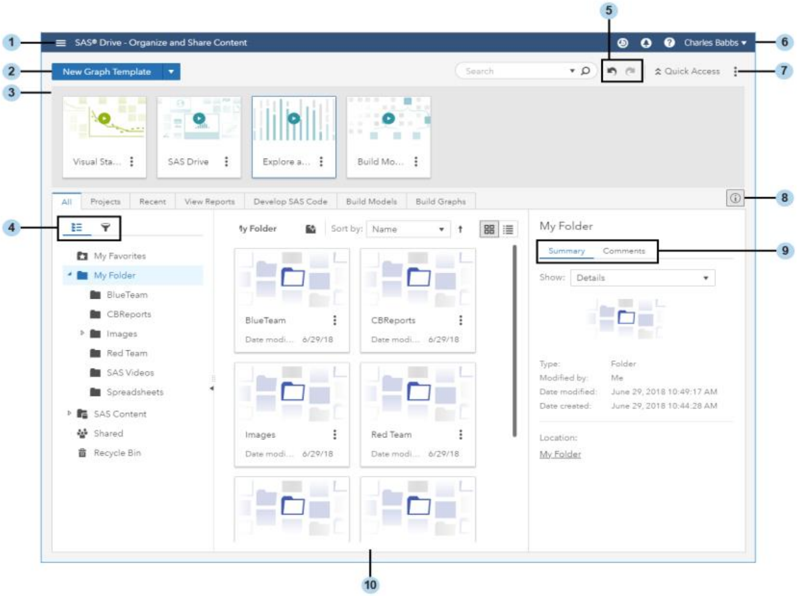
8

The “heart” of the SAS Platform is the analytics life cycle, and the real value from machine learning is derived with the actionable insights throughout the analytics life cycle. In its simplest form, the analytics life cycle is a series of activities with the goal to extract value from raw data. The definition of “value” is specific to your organization’s goals and objectives.

The three phases of the analytics life cycle are ***data***, ***discovery***, and ***deployment***. Data are the foundation of everything you do, discovery is the act of finding something that you had not known before, and deployment is where you get the value out of analytics. Recognizing and fully supporting all three is necessary to generate impactful insights that come from transforming data into value. You can start in any of them, but it is important to know the next step and how to get there.

SAS delivers the ability to address and connect each phase of this analytics life cycle. SAS is uniquely positioned to offer the complete picture.

SAS Drive



SAS Drive is a hub for the SAS Viya applications, and it enables you to easily view, organize, and share your content from one place.

SAS Drive uses the standard sign-in window for SAS applications. To display a sign-in window, enter the URL provided by your administrator (for example, **https://***prod.host.com***/SASDrive**).

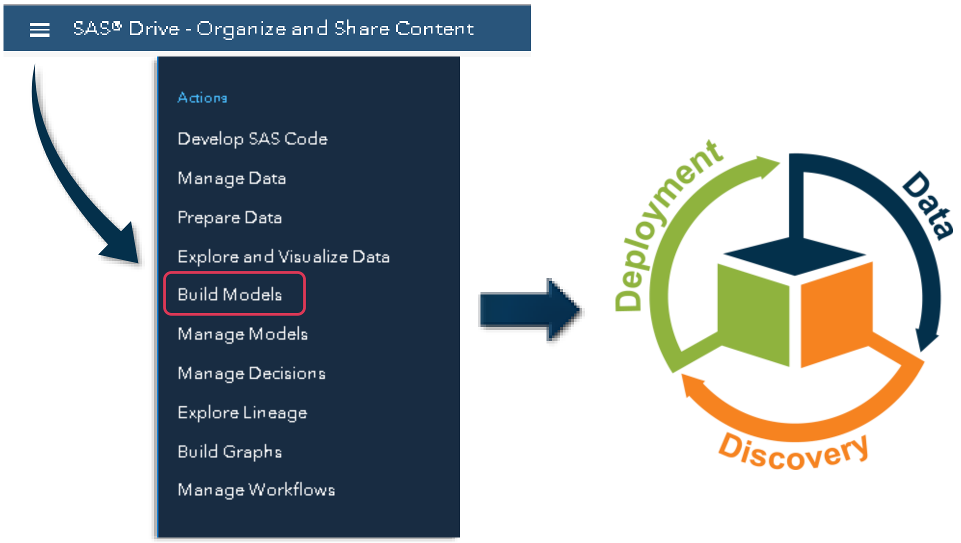
SAS Drive is always available from the Applications menu in the upper left.

1. Applications menu.
2. New item button.
3. Quick Access area.
4. Folders and Filter.
5. Undo and Redo. Click and hold on either icon to display a list of actions.
6. Recent items, Notifications, Help, Settings, and Sign out.
7. Menu. Create links or shortcuts, manage tabs, upload content.
8. Information pane button.
9. Summary and Comments tabs.
10. Canvas.

The displayed tabs depend on the products that are installed at your site.

My Folder is a shortcut to **/SAS Content/Users/[userID]/MyFolder/**.

Common Interface for Entire Analytics Life Cycle



SAS® Drive - Organize and Share Content

Actions

- Develop SAS Code
- Manage Data
- Prepare Data
- Explore and Visualize Data
- Build Models
- Manage Models
- Manage Decisions
- Explore Lineage
- Build Graphs
- Manage Workflows

10

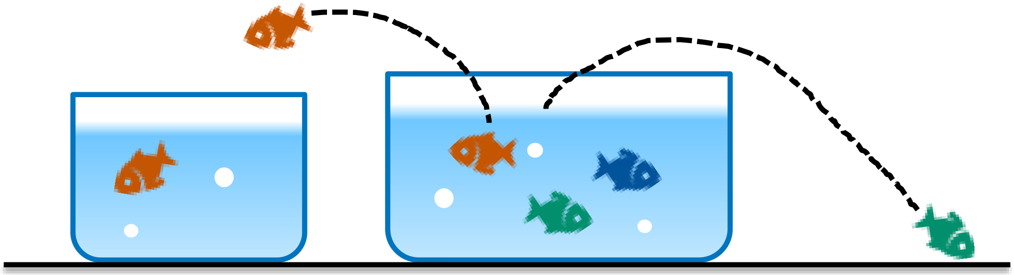
SAS Drive is a common interface for the SAS Viya applications that supports all three phases of the analytics life cycle.

The availability of the features in SAS Drive depends on the applications that have been installed and the features and permissions that have been specified by your administrator.

Note: This course focuses on the Build Models action that launches Model Studio pipelines. Model Studio and pipelines are discussed soon.

Business Challenge: Customer Churn

Customer churn, also known as *customer attrition*, is when an existing customer, subscriber, user, or any return client stops doing business or ends the relationship with a company.



11

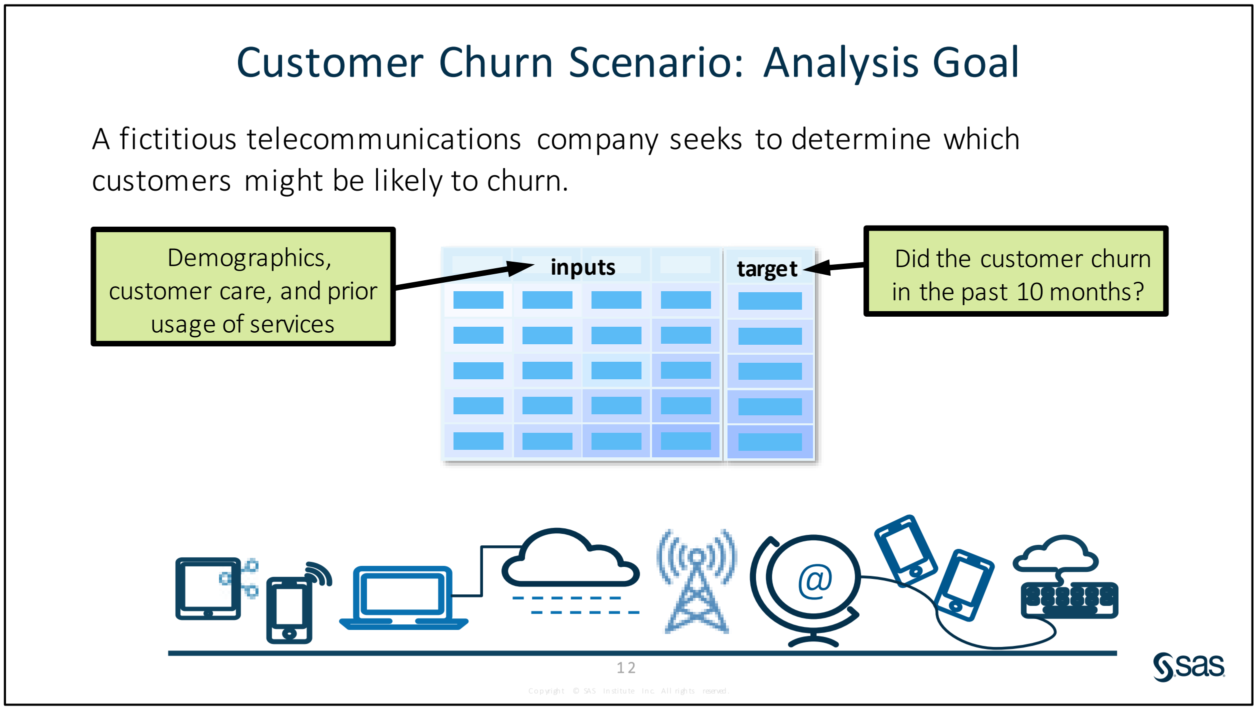
This course emphasizes churn. Nevertheless, the approach shown here can be applied to any business problem regardless of the domain.

Customer churn is one of the main problems in many businesses. Studies have shown that attracting new customers is much more expensive than retaining existing ones. Consequently, companies focus on developing accurate and reliable predictive models to identify potential customers who will churn soon.

Analysis of churn prediction covers several phases:

- understanding the business
- selection, analysis, and data processing
- implementing various algorithms for classification
- evaluating the classifiers and choosing the best one for prediction

The obtained results should be of great value for management.



Customer Churn Scenario: Analysis Goal

A fictitious telecommunications company seeks to determine which customers might be likely to churn.

Demographics, customer care, and prior usage of services

Did the customer churn in the past 10 months?

The communications sector evolves and grows more than virtually any other industry. Mobile devices and broadband connectivity continue to be more embedded in every aspect of society, driving how we live and work, including trends in video streaming, Internet of Things, mobile payments, messaging, geo-localization, and social media. This growth comes with new players, representing opportunities but also threats. And churn is now beyond the traditional barriers. Companies need to understand consumers' behaviors to deploy effective retention and loyalty programs.

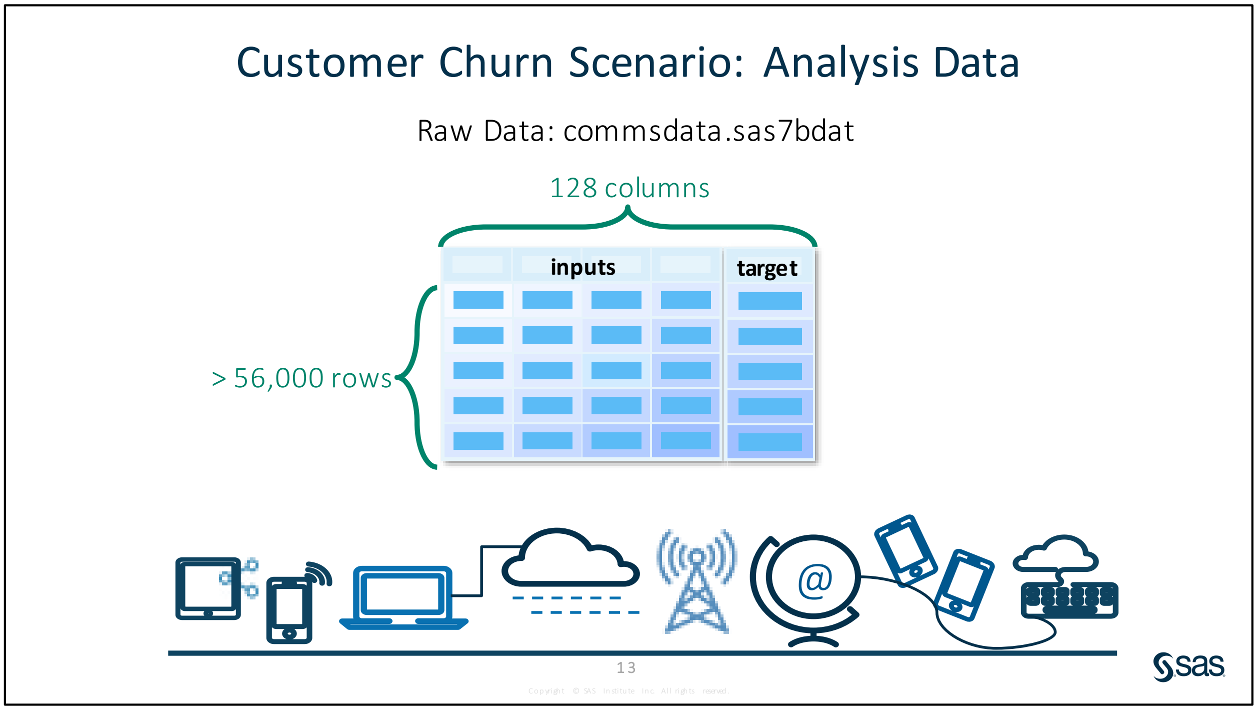
The analytics life cycle starts with defining the analysis goal regarding the business problem at hand. The business case study in this course refers to a fictitious telecommunications company, which is trying to reduce its churn rate. Churn is an inescapable reality in the telecom world. Customers come and go, swayed by the latest disappointments or the latest deals. This churn is costly. It is far more expensive to acquire a customer than to satisfy and retain an existing one.

Analytics has proven its value in identifying customers at risk for churn and helping marketers understand how to retain them. Major service providers have adopted churn propensity models that have succeeded in reducing the churn rate for post-paid subscribers in an effective way. This fictitious telecommunications company has a reasonable amount of data to describe the customer's behavior. The main goal is to use this data and train supervised learning models to predict the churn event.

Other Machine Learning Applications in Telecommunications

Other machine learning applications in the telecommunication industry include network management and network optimization. This includes modeling and predicting network traffic fluctuations to optimize quality of service and routing by being able to reallocate bandwidth as needed, identify and resolve network bottlenecks, manage capacity to plan for infrastructure investments, maintain quality of service, and optimize the network for their most valuable customers.

Fraud detection is another example in the telecommunication sector. Predictive models are built to protect customers and their bottom line by proactively detecting fraudulent activities. You utilize usage data, location-specific data, and customer account data in real time to model baseline "normal" behavior and flag when behavior deviates from this "normal." Atypical phone calls that might indicate theft or hacking can be flagged.



Customer Churn Scenario: Analysis Data

Raw Data: commsdata.sas7bdat

128 columns

inputs

target

> 56,000 rows

13

Copyright © SAS Institute Inc. All rights reserved.

The **commsdata** data set contains a reasonable amount of data that describe consumer behavior. The input variables include demographic type variables, variables that describe product usage and type, billing data, and customer service/call center information. The main goal is to use this set of input variables and train supervised learning models to predict the churn event.

ID variable

Name	Label	Description
Customer_ID	Primary Key	Unique identification of the customer.

Target variables

Name	Label	Description
Churn	Churn	Indicates whether customers churned.
Upsell_xsell	Xsell Upsell Flag	Indicates customer's flag for cross-sell or up-sell. (You do not use this variable in this course.)

Categorical-valued inputs

Name	Label	Description
credit_class	Credit Class	Credit category for an account or customer. It summarizes the overall credit worthiness of a customer or account.
sales_channel	Acquisition Channel	The way consumer was persuaded to purchase company's services.
region	Account Region	Customer account region.
state	Account State	Customer state location.
city	Account City	City designation for customer address.
zipcode_primary	Account Code	Primary customer ZIP code.
product_plan_desc	Plan Name	Customer's product plan.
handset_age_grp	Handset Age Group	Customer's handset age in days.
handset	Handset Mfg	Handset manufacturer. Values include *Apple*, *HTC*, *LG*, *Motorola*, *Nokia*, *Samsung*, and *Unknown*.
lifestage	Plan Life Stage	Type of contract.
rp_pooled_ind	Pooled Rate Plan	Indicates whether customer has pooled rate.
call_center	Last Call Center Used	Location of the last call center used.
issue_level1	Call Center Issue Level 1	Level 1 reason of the call.
issue_level2	Call Center Issue Level 2	Level 2 reason of the call.
call_category_1	Call Center Category 1	Category 1 for the call.
call_category_2	Call Center Category 2	Category 2 for the call.
resolution	Final Resolution	Resolution action taken by call center.
verbatims	Survey Verbatim	Feedback from customers via call centers.

Interval-valued inputs

Name	Label	Description
lifetime_value	Lifetime Value	Customer's value.
avg_arpu_3m	3M Avg Revenue per User	Average revenue for the past three months.
acct_age	Account Tenure	Number of months that the account has been active.
billing_cycle	Billing Cycle	Customer's billing cycle (period of the month).
nbr_contract_ltd	Total Number Contract lifetime	Number of contracts during life cycle.
rfm_score	Account Ranking (RFM Score)	Customer's account score.
Est_HH_Income	Estimated HH Income	Household income.
region_lat	Account Region Latitude	Customer region latitude.
region_long	Account Region Longitude	Customer region longitude.
state_lat	Account State Latitude	State latitude.
state_long	Account State Longitude	State longitude.
city_lat	Account City Latitude	Customer city latitude.
city_long	Account City Longitude	Customer city longitude.
zip_lat	Account ZIP Code Latitude	ZIP code latitude.
zip_long	Account ZIP Code Longitude	ZIP code longitude.
cs_med_home_value	Census Area Median Home Value Index	Median home value in customer's area.
cs_pct_home_owner	Census Area Percent Home Owner	Percentage home owner in customer's area.
cs_ttl_pop	Census Area Total Population	Population in customer's area.
cs_hispanic	Census Area Hispanic	Hispanic population in customer's area.
cs_caucasian	Census Area Caucasian	Caucasian population in customer's area.
cs_afr_amer	Census Area African-American	African-American population in customer's area.
cs_other	Census Area Other	Other population in customer's area.
cs_ttl_urban	Census Area Total Urban	Urban population in customer's area.

Name	Label	Description
cs_ttl_rural	Census Area Total Rural	Rural population in customer's area.
cs_ttl_male	Census Area Total Males	Male population in customer's area.
cs_ttl_female	Census Area Total Females	Female population in customer's area.
cs_ttl_hhlds	Census Area Total Households	Households in customer's area.
cs_ttl_mdage	Census Area Median Age	Median age in customer's area.
mb_inclplan	Plan Data MB	MB included in the plan.
ever_days_over_plan	Total Days Over Plan	Total days over the plan.
ever_times_over_plan	Total Times Over Plan	Total times over the plan.
data_device_age	Avg Age of Devices on Plan	Average age of devices on the plan.
equip_age	Handset Age	Age of equipment history, whether mobile device, smartphone, or another handset type.
mfg_apple	Own Apple	Apple manufactured device. 1 is *Yes*, 0 means *No*.
mfg_samsung	Own Samsung	Samsung manufactured device. 1 is *Yes*, 0 means *No*.
mfg_htc	Own HTC	HTC manufactured device. 1 is *Yes*, 0 means *No*.
mfg_motorola	Own Motorola	Motorola manufactured device. 1 is *Yes*, 0 means *No*.
mfg_lg	Own LG	LG manufactured device. 1 is *Yes*, 0 means *No*.
mfg_nokia	Own Nokia	Nokia manufactured device. 1 is *Yes*, 0 means *No*.
delinq_indicator	Delinquent Indicator	Delinquency indicator. Scale varies from -2 to +4, depending on customer history.
times_delinq	Consecutive Mths Delinquent	Consecutive months in default.
count_of_suspensions_6m	Times Suspended Last 6M	Times suspended in the past six months.
avg_days_susp	Days Suspended Last 6M	Days suspended in the past six months.
calls_total	Total Calls Curr	Current number of calls.

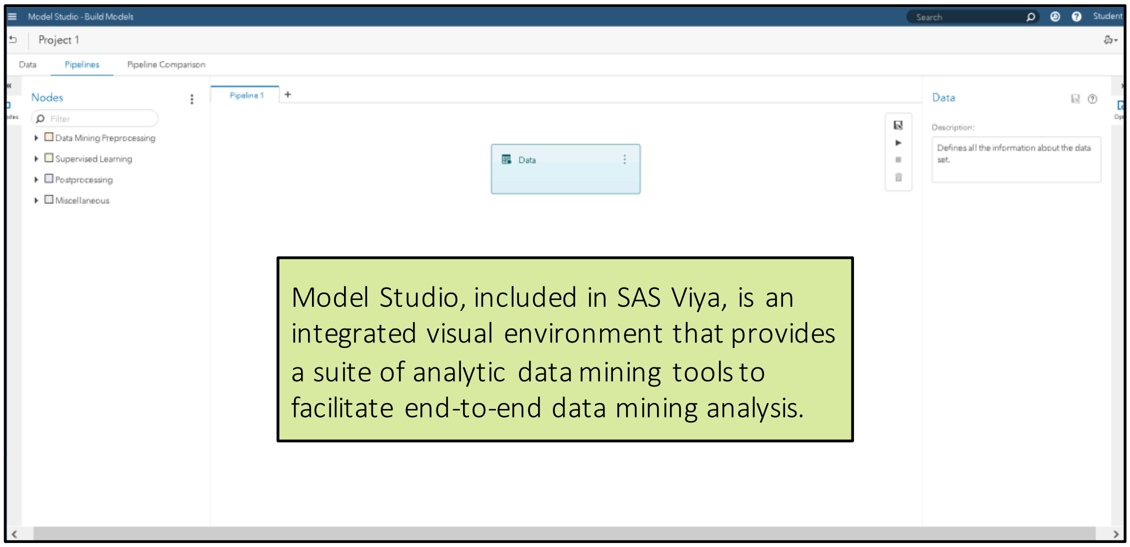
Name	Label	Description
calls_in_pk	Calls Incoming Peak	Number of calls received in peak time.
calls_in_offpk	Calls Incoming Off-Peak	Number of call received off peak time.
calls_out_offpk	Calls Outgoing Off-Peak	Number of calls made in peak time.
calls_out_pk	Calls Outgoing Peak	Number of calls made off peak time.
mou_total_pct_MOM	Minutes Total Pct Change Month over Month	Percentage of minutes change month over month.
mou_onnet_pct_MOM	Minutes on Network Pct Change Month over Month	Percentage of minutes on network change month over month.
mou_roam_pct_MOM	Minutes Roaming Pct Change Month over Month	Percentage of minutes on roaming change month over month.
mou_onnet_6m_normal	6M Avg Minutes on Network Normally Distributed	Minutes of use on network over six months normally distributed.
mou_roam_6m_normal	6M Avg Minutes Roaming Normally Distributed	Minutes of use in roaming over six months normally distributed.
voice_total_bill_mou_curr	Total Voice Billed Minutes of Use	Current minutes of voice billed.
tot_voice_chrgs_curr	Total Voice Charges	Current minutes of voice charged.
tot_drpd_pr1	Number of Dropped Calls 1 Mth Prior	Number of dropped calls on the previous month.
bill_data_usg_m03	3M Avg Billed Data Usage	Average data billed over the past three months.
bill_data_usg_m06	6M Avg Billed Data Usage	Average data billed over the past six months.
bill_data_usg_m09	9M Avg Billed Data Usage	Average data billed over the past nine months.
mb_data_usg_m01	MB Data Usage 1 Mth Prior	MB data used on the previous month.
mb_data_usg_m02	MB Data Usage 2 Mths Prior	MB data used prior two months.
mb_data_usg_m03	MB Data Usage 3 Mths Prior	MB data used prior three months.
mb_data_ndist_mo6m	6M Avg Billed Data Usage Normally Distributed	Data used on network over six months normally distributed.

Name	Label	Description
mb_data_usg_roamm01	MB Data Usage Roam 1 Mth Prior	Data used in roaming in the previous month.
mb_data_usg_ roamm02	MB Data Usage Roam 2 Mths Prior	Data used in roaming prior two months.
mb_data_usg_ roamm03	MB Data Usage Roam 3 Mths Prior	Data used in roaming prior three months.
data_usage_amt	Data Usage Amount	Total data usage amount over last month.
tweedie_adjusted	Data Usage Amt Tweedie Distributed	Data used in Twitter.
tot_mb_data_curr	Total MB of Data Usage	Current MB data used.
tot_mb_data_roam_curr	Total MB of Roam Data Usage	Current MB data used in roaming.
bill_data_usg_total	Total Billed Data usage	Total billed data.
tot_overage_chgs	Total Overage Charges	Total overage charged.
data_prem_chrgs_curr	Premium Data Charges	Premium data charged.
nbr_data_cdrs	Number of Data Records	Number of call detail records.
avg_data_chrgs_3m	3M Avg Data Charges	Average data charged in the past three months.
avg_data_prem_chrgs_3m	3M Avg Premium Data Charges	Average premium data charged in the past three months.
avg_overage_chrgs_3m	3M Avg Overage Charges	Average overage data charged in the past three months.
nbr_contacts	Number Times Customer Contacted	Number of contacts customer made to the company.
calls_TS_acct	Number Calls Tech Support	Number of tech support calls.
open_tsupcomplnts	Open Tech Support Complaints	Number of tech support complains opened.
num_tsupcomplnts	Tech Support Complaints - LTD	Number of tech support complains.
unsolv_tsupcomplnts	Unresolved Tech Support Complaints - LTD	Number of unsolved tech support complaints.
wrk_orders	Open Work Orders	Number of open work.
days_openwrkorders	Days of Open Work Orders	Days of open work.
resolved_complnts	Resolved Complaints	Number of complaints resolved.

Name	Label	Description
calls_care_acct	Number Calls Care Center	Call center care account assignment, which takes values between 0-9.
calls_care_3mavg_acct	Number Calls Care Center 3 Month Avg	Call center care account score over past three months averaged.
calls_care_6mavg_acct	Number Calls Care Center 6 Month Avg	Call center care account score over past six months averaged.
res_calls_3mavg_acct	Resolved Calls – 3Mo Average	Average number of resolved customer service calls over past three months for the customer account.
res_calls_6mavg_acct	Resolved Calls – 6Mo Average	Average number of resolved customer service calls over past six months for the customer account.
last_rep_sat_score	Last Call Satisfaction Rating Given	Latest customer service representative satisfaction score (given by past customers).
network_mention	Network Issues Discussed	Number of network issues discussed.
service_mention	Service Issues Discussed	Number of service issues discussed.
price_mention	Price Issues Discussed	Number of prices issues discussed.
times_susp	Number of Times Suspended	Number of times suspended.
curr_days_susp	Number of Days Suspended	Number of days suspended.
pymts_late_ltd	Total Late Payments Lifetime	Number of late payments.
calls_care_ltd	Total Calls to Care Lifetime	Number of calls to call center.
MB_Data_Usg_M04	MB of Data Usage Month 4	MB data used in past four months.
MB_Data_Usg_M05	MB of Data Usage Month 5	MB data used in past five months.
MB_Data_Usg_M06	MB of Data Usage Month 6	MB data used in past six months.
MB_Data_Usg_M07	MB of Data Usage Month 7	MB data used in past seven months.
MB_Data_Usg_M08	MB of Data Usage Month 8	MB data used in past eight months.

Name	Label	Description
MB_Data_Usg_M09	MB of Data Usage Month 9	MB data used in past nine months.
seconds_of_data_norm	Seconds of Data - Normalized	Number of seconds of data normalized.
seconds_of_data_log	Seconds of Data - Natural Log	Number of seconds of data transformed by log.

Model Studio



Model Studio, included in SAS Viya, is an integrated visual environment that provides a suite of analytic data mining tools to facilitate end-to-end data mining analysis.

14

Model Studio enables you to explore ideas and discover insights. In other words, it is part of discovery piece of the analytics life cycle. Model Studio is a central, web-based application that includes a suite of integrated data mining tools. The data mining tools supported in Model Studio are designed to take advantage of the SAS Viya programming and cloud processing environments to deliver and distribute analytic model data mining champion models, score code, and results.

Model Studio is a common interface that contains the following SAS solutions:

- SAS Visual Forecasting
- SAS Visual Data Mining and Machine Learning in Model Studio
- SAS Visual Text Analytics

This course addresses the following SAS platform components:

Component	Component Name
Foundation	SAS Viya
Software Product	SAS Visual Data Mining and Machine Learning
SAS Drive Application Shortcut	Build Models
Interface	Model Studio / Pipelines

Note: The visual analytic data mining tools that appear in Model Studio are determined by your site's licensing agreement. Model Studio operates with one, two, or all three of the web-based analytic tools as components of the software.

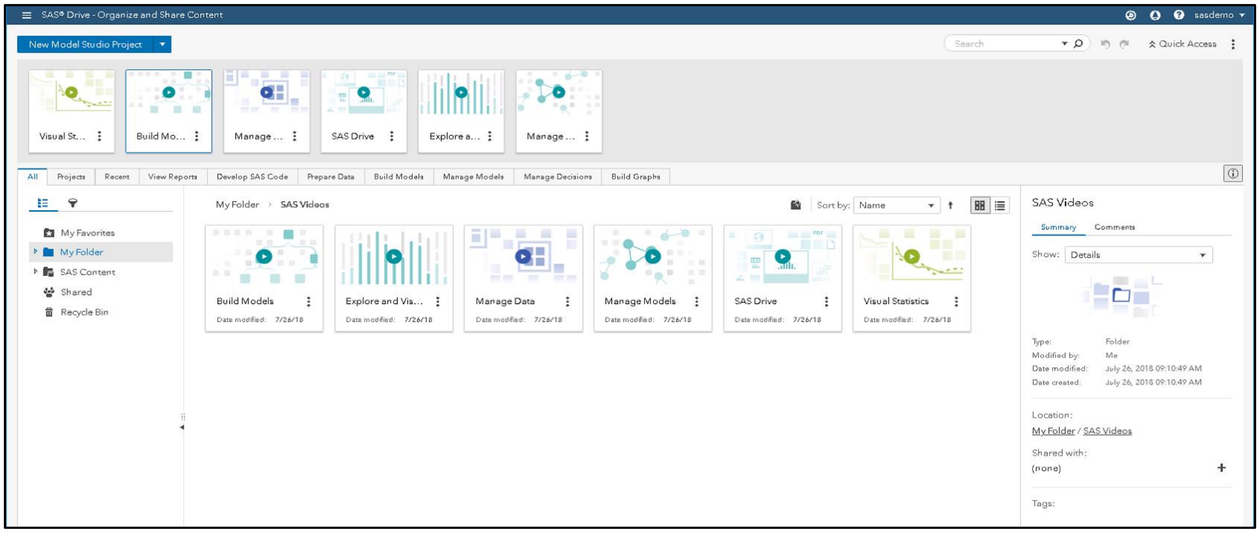
Note: Model Studio comes with SAS Data Preparation Basic. SAS Data Preparation is a software offering that adds data quality transformations and other advanced features. There are several options that enable you to perform specific data preparation tasks for applications, such as SAS Environment Manager, SAS Visual Analytics, Model Studio, and SAS Decision Manager. You can perform some of the basic data preparation tasks through Model Studio.

Creating a Project and Loading Data

In this demonstration, you create a new project in Model Studio based on the **commsdata** data set. A project is a top-level container for your analytic work in Model Studio. The table will be imported from a local drive. The type of project will be defined. This project will be used to predict churn for a fictitious telecommunications company. A target variable will be selected for this table.

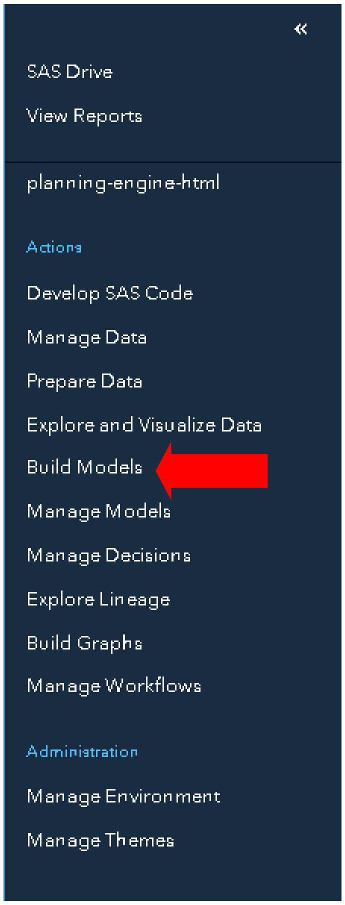
1. From the Windows taskbar, launch Google Chrome. When the browser opens, select **SAS Viya** ⇨ **SAS Drive** from the bookmarks bar or from the link on the page.
2. The user ID and password should be pre-filled. If not, use the following:
   a. Enter **student** in the **User ID** field.
   b. Enter **Metadata0** in the **Password** field.

   Note: Use caution when you enter the user ID and password because values can be case sensitive.
3. Click **Sign In**.
4. Select **Yes** in the Assumable Groups window. The SAS Drive home page appears.

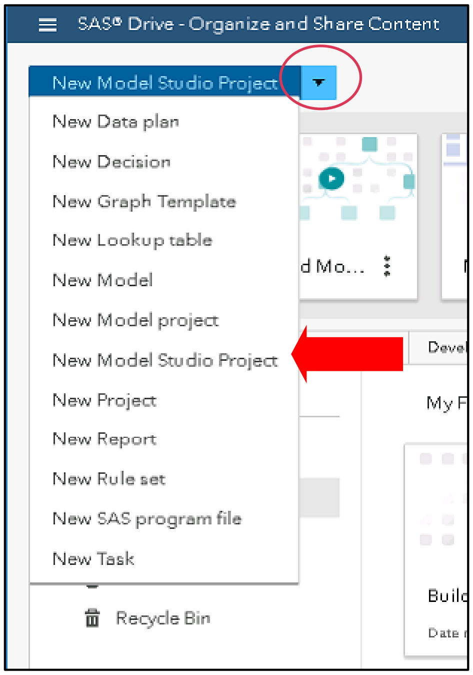


Note: The SAS Drive page on your classroom computer might not have the same tiles as the image above.

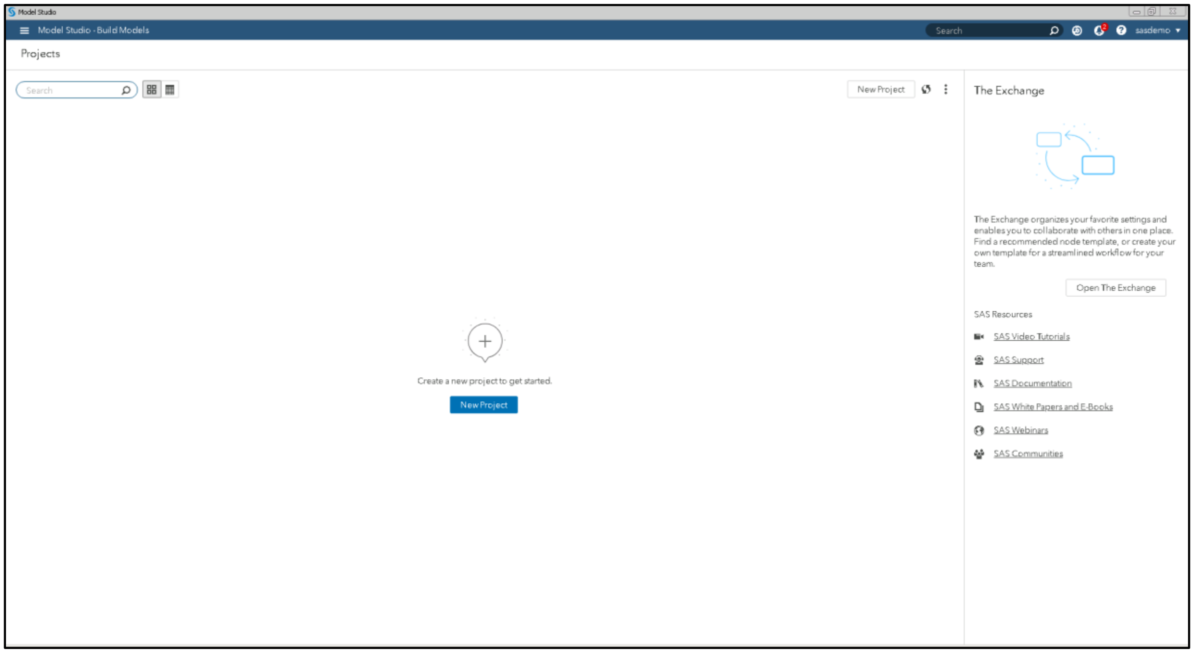
5. Click the **Applications** menu (≡) in the upper left corner of the SAS Drive page. Select **Build Models**.



Alternatively, click the menu in the upper left corner to create a new item. Select **New Model Studio Project** from the menu.



The Model Studio Projects page is now displayed.



The Exchange organizes your favorite settings and enables you to collaborate with others in one place. Find a recommended node template, or create your own template for a streamlined workflow for your team.

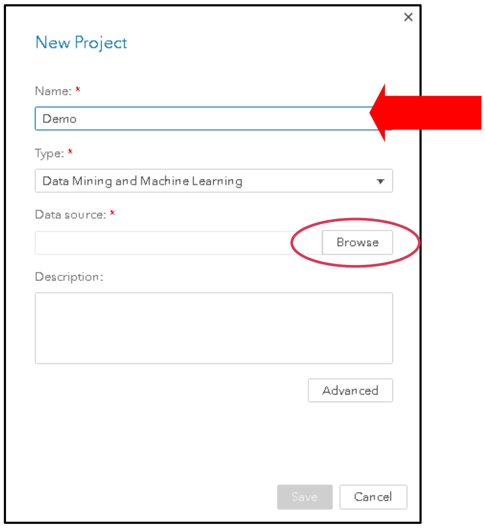
Note: The Projects page might differ on your classroom computer from the image above. There might be pre-existing projects on your classroom computer.

From the Model Studio Projects page, you can view existing projects, create new projects, and access the Exchange. Model Studio projects can be one of three types (depending on the SAS licensing for your site): Forecasting projects, Data Mining and Machine Learning projects, and Text Analytics projects.

Note: The Exchange organizes your favorite settings and enables you to collaborate with others in one place. Find a recommended node template or create your own template for a streamlined workflow for your team. The Exchange is accessed later in the course,

6. Select **New Project** in the upper right corner of the Projects page.
7. Enter **Demo** as the name in the New Project window. Leave the default Type of **Data Mining and Machine Learning**.

   Select **Browse** in the Data Source field.



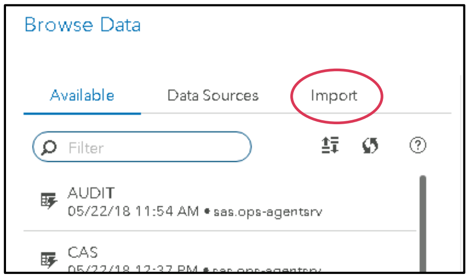
New Project

Name: * Demo

Type: * Data Mining and Machine Learning

8. Import a SAS data set into CAS.

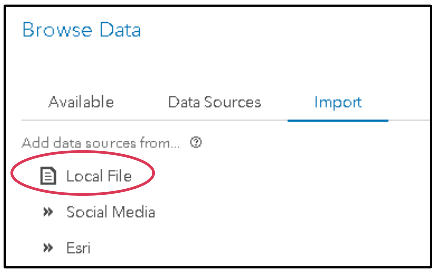
   a. In the Browse Data window, click **Import**.



Browse Data

Available
Data Sources
Import
Filter
AUDIT
05/22/18 11:54 AM • sas.ops-agentsrv
CAS
05/22/18 12:37 PM • sas.ops-agentsrv

   b. Under Import, select **Local File**.

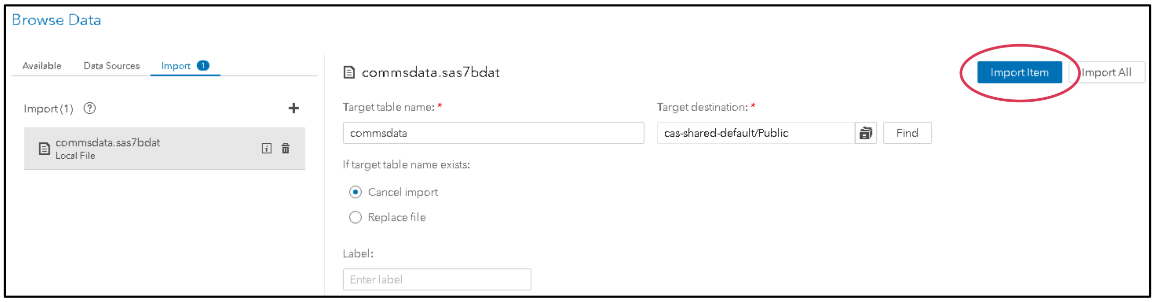


Browse Data

   c. Navigate to **D:\Workshop\Winsas\CPML**.

   d. Select the **commsdata.sas7bdat** table. Click **Open**.

e. Select **Import Item**. Model Studio parses the data set and pre-populates the window with data set configurations.



Target table name: * commsdata

Target destination: * cas-shared-default/Public

If target table name exists:

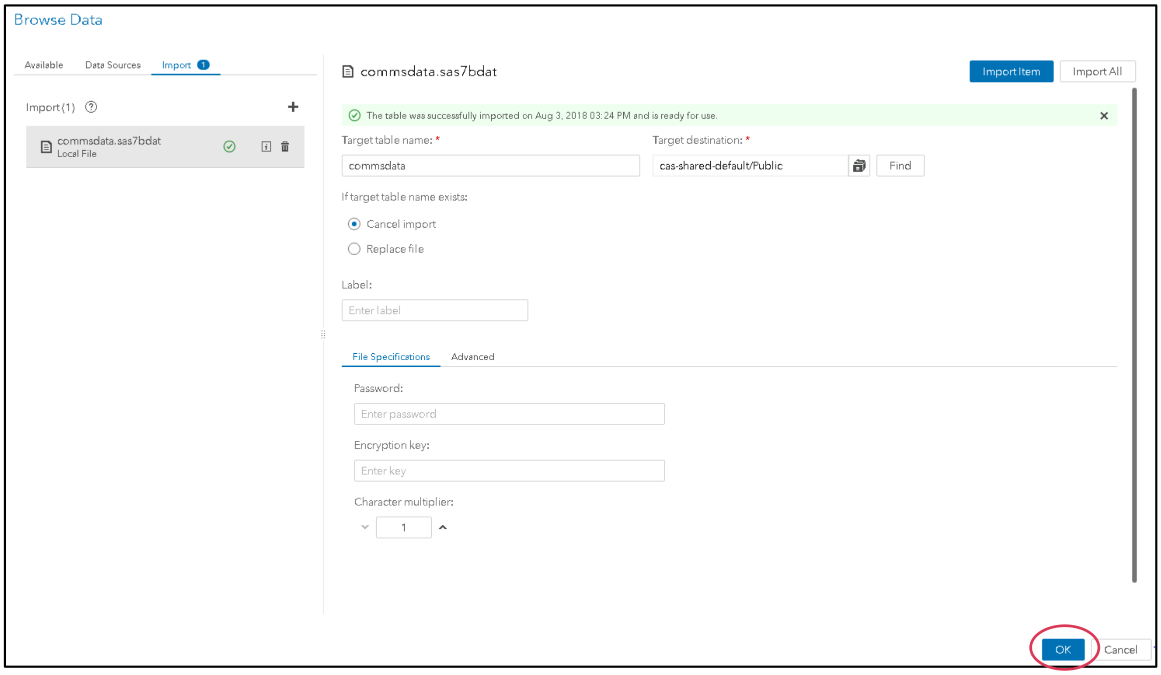
● Cancel import

○ Replace file

Label: Enter label

Note: When the data is in memory, it is available for other projects through the Available tab.

f. Click **OK** after the table is imported.



The table was successfully imported on Aug 3, 2018 03:24 PM and is ready for use.

Target table name: * commsdata

Target destination: * cas-shared-default/Public

If target table name exists:

● Cancel import

○ Replace file

Label: Enter label

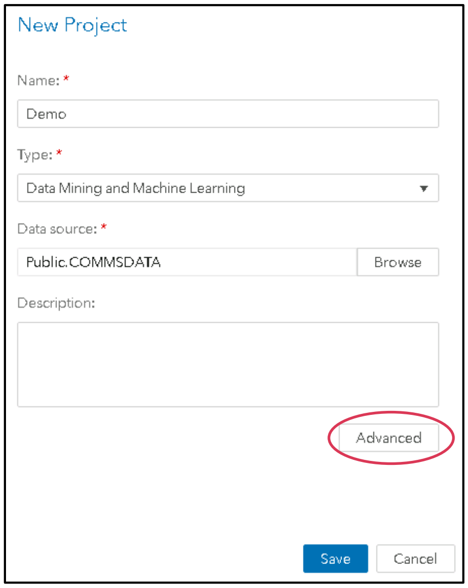
Password: Enter password

Encryption key: Enter key

Character multiplier: 1

Note: Tables are imported into the CAS server and are available to use with SAS Visual Analytics. When the import is complete, you are returned to Model Studio. For more information about data types supported in CAS and how to load data into CAS, see the details section at the end of this demo.

9. Click **Advanced** in the New Project window.



New Project

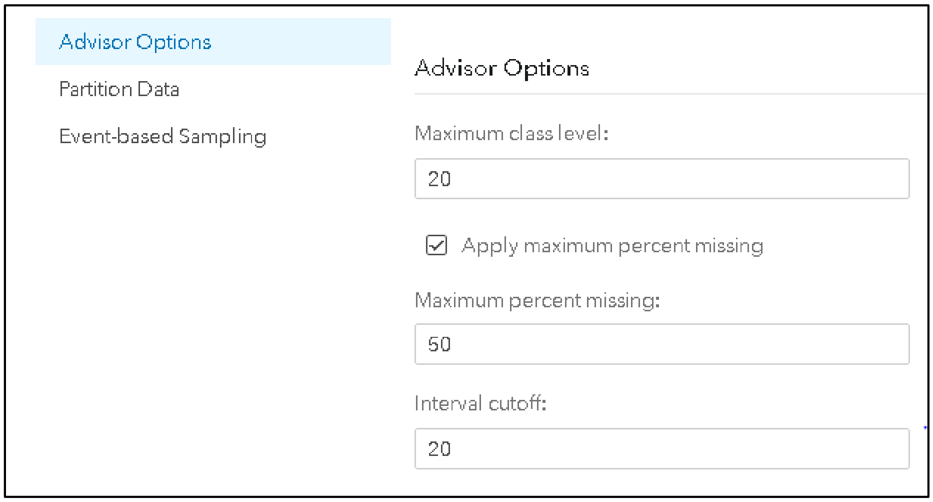
Name: * Demo

Type: * Data Mining and Machine Learning

Data source: * Public.COMMSDATA

Description:

10. The Advanced project settings appear, and **Advisor Options** is one of the selection. The Advisor Options section enables you to define the threshold for interval/nominal variables (if a numeric input has more levels than the ***interval cutoff***, it will be interval; otherwise, it will be nominal), the threshold to reject categorical variables (if a nominal input has more than the ***maximum class level***, then it will be rejected) and the threshold to reject missing variables (if a variable has more missing values than the ***maximum percent missing***, it will be rejected).



Advisor Options

Maximum class level: 20

☑ Apply maximum percent missing

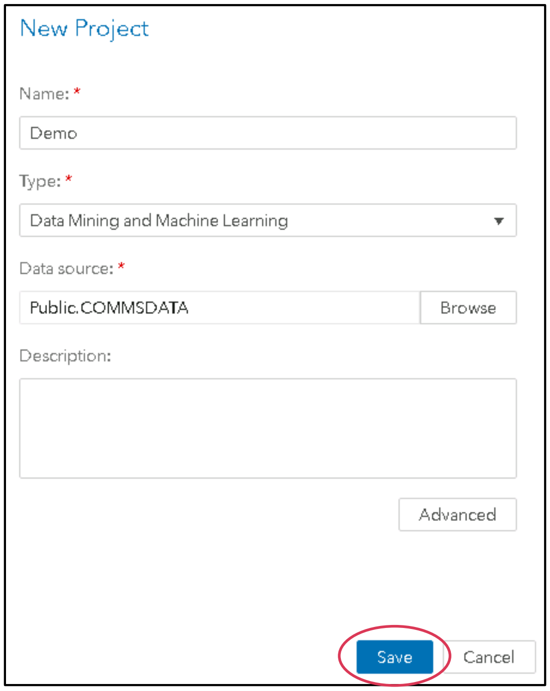
Maximum percent missing: 50

Interval cutoff: 20

The Advanced settings also include **Partition Data** and **Event-based Sampling** options, but these topics are covered in the next section, and they are discussed in the next demonstration. In addition to accessing Partition Data and Event-based Sampling options here, they can also be accessed after the project is created.

Click **Cancel** to return to the New Project window.

11. Click **Save**.



New Project

Name: *

Demo

Type: *

Data Mining and Machine Learning

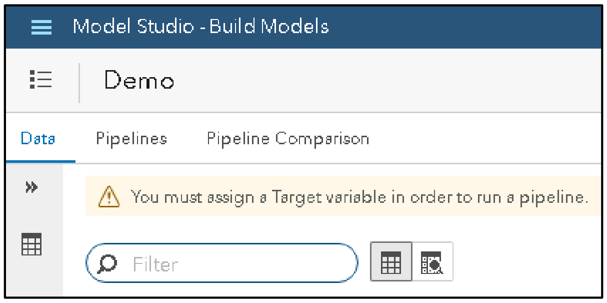
Data source: *

Public.COMMSDATA

Description:

Note: After you create your new project, Model Studio takes you to the Data tab of your new project page. Here, you can adjust data source variable names, labels, type, role, and level assignments. The Data tab enables you to modify variable assignments and manage global metadata. You can also retrain a model with new data, if the target variable in the new data set is the same as the original data set.

12. When the project is created, you need to assign a target variable in order to run a pipeline.



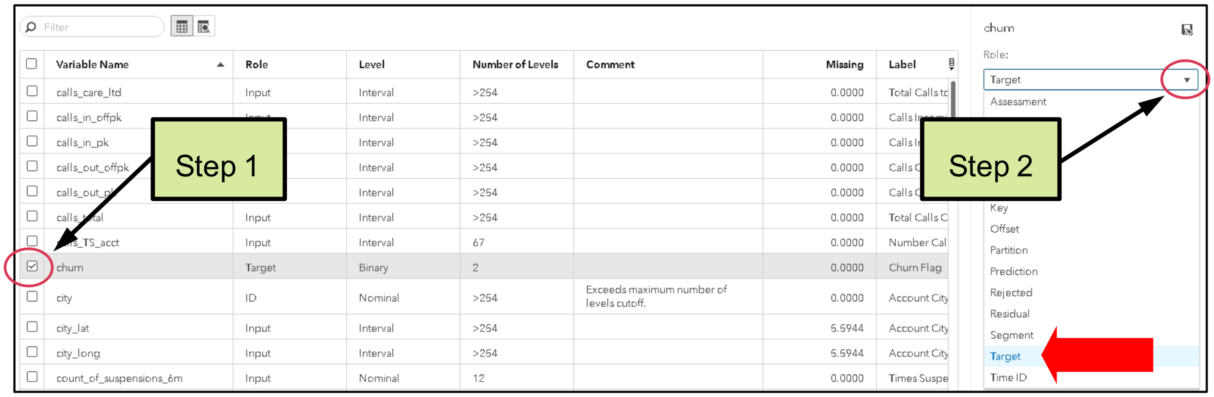
Model Studio - Build Models

Demo

You must assign a Target variable in order to run a pipeline.

You can also have target variable role already defined in your data. Model Studio provides several options for managing and modifying data. The Data tab enables you to modify variable assignments and manage global metadata. You can also retrain a model with new data, if the target variable in the new data set is the same as the original data set.

13. In the variables window, select **churn** (Step 1). Then in the right pane, select **Target** under the **Role** property (Step 2).



□	Variable Name	Role	Level	Number of Levels	Comment	Missing	Label
□	calls_care_ltd	Input	Interval	>254		0.0000	Total Calls tc
□	calls_in_offpk	Input	Interval	>254		0.0000	Calls Incomi
□	calls_in_pk		Interval	>254		0.0000	Calls I
□	calls_out_offpk		Interval	>254		0.0000	Calls C
□	calls_out_pk		Interval	>254		0.0000	Calls C
□	calls_total	Input	Interval	>254		0.0000	Total Calls C
□	calls_TS_acct	Input	Interval	67		0.0000	Number Cal
☑	churn	Target	Binary	2		0.0000	Churn Flag
□	city	ID	Nominal	>254	Exceeds maximum number of levels cutoff.	0.0000	Account City
□	city_lat	Input	Interval	>254		5.5944	Account City
□	city_long	Input	Interval	>254		5.5944	Account City
□	count_of_suspensions_6m	Input	Nominal	12		0.0000	Times Suspe

End of Demonstration

Details: CAS-Supported Data Types and Loading Data into CAS

Caslibs

All data is available to the CAS server through caslibs, and all operations in the CAS server that use data are performed using caslibs. A caslib provides access to files in a data source, such as a database or a file system directory, and to in-memory tables. Access controls are associated with caslibs to manage access to data. You can think of a caslib as a container where the container has two areas where data is referenced: a physical space that includes the source data or files, and an in-memory space that makes the data available for CAS action processing. Authorized users can add or manage caslibs with the CASLIB statement. Caslib authorization is set by your administrator. In some instances, such as when you copy native CAS tables that are not in-memory, a caslib is required although data is not copied to the caslib.

Load Data to a Caslib

You can load a SAS data set, database tables, and more to a caslib. DATA step, the CASUTIL procedure, and CAS actions can be used to load data in to CAS. After the data is in a caslib, you can use a DATA step, procedures, CAS actions, PROC DS2, or PROC FEDSQL operations on the CAS table. Tables are not automatically saved when they are loaded to a caslib. You can use PROC CASUTIL to save tables. Native CAS tables have the file extension .sashdat.

Note: For information about file types that are supported in CAS, see "Path-Based Data Source Types and Options" in *SAS® Cloud Analytic Services: User's Guide* and the free video tutorial "Understanding Caslibs and Loading Data in SAS Viya" available here: https://video.sas.com/category/videos/an-introduction-to-sas-viya-programming-for-sas-9-programmers

Data Types

The CAS server supports the VARCHAR, INT32, INT64, and IMAGE data types in addition to the CHARACTER and NUMERIC data types, which are traditionally supported by SAS.

Variables that are created using the VARCHAR data type are varying width and use character semantics, rather than being fixed-width and using the byte semantics of the traditional CHARACTER data type. Using the VARCHAR data type in the DATA step in the CAS server has some restrictions.

Note: For more information, see "Restrictions for the VARCHAR Data Type in the CAS Engine" in *SAS® Cloud Analytic Services: User's Guide*.

Variables that are created or loaded using the INT32 or INT64 data types support more digits of precision than the traditional NUMERIC data type. All calculations that occur on the CAS engine maintain the INT32 or INT64 data type. Calculations in DATA steps or procedures that run on the SAS®9 engine are converted to NUMERIC values.

The CHARACTER and NUMERIC data types continue to be the supported data types for processing in the SAS Workspace Server.

The DS2 language supports several additional data types. On the CAS server, DS2 converts non-native data types to CHARACTER, NUMERIC, or VARCHAR.

Note: For information about data types that are supported for specific data sources, see "Data Type Reference" in *SAS® DS2 Language Reference*.

The CAS language (CASL) determines the data type of a variable when the variable is assigned.

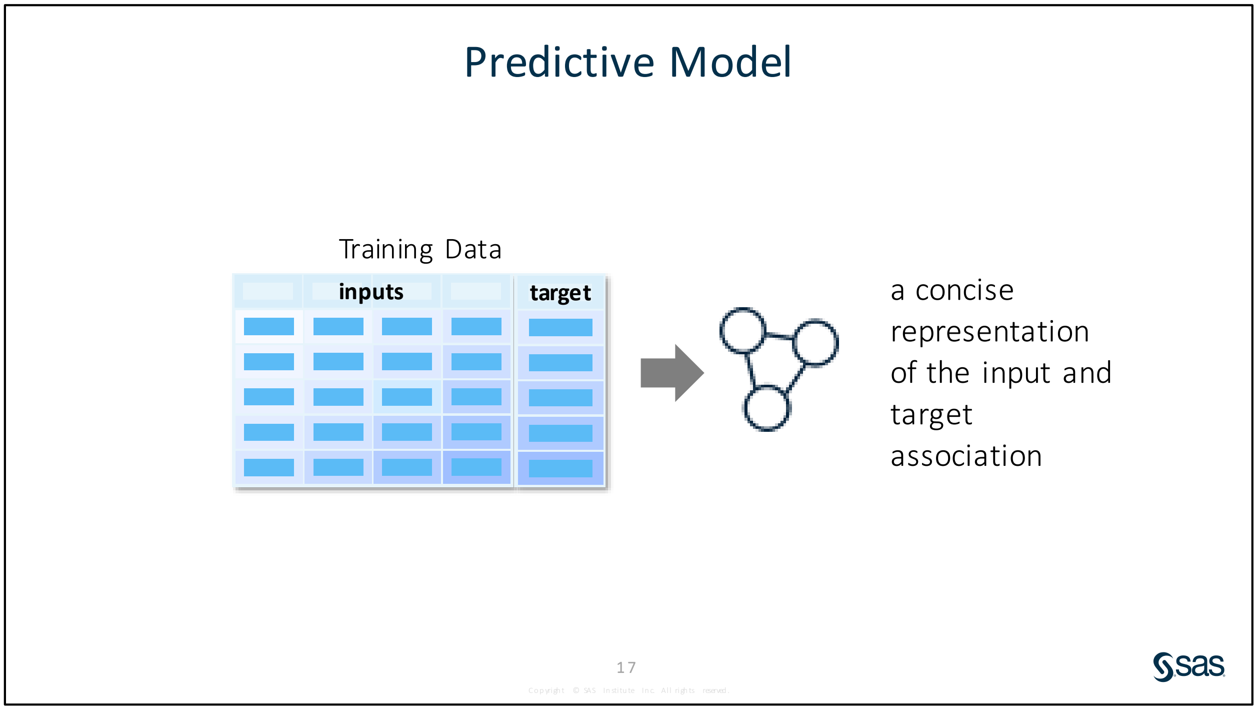
In the following table, the letter Y indicates the data types that are supported for programming on the CAS server. In the last column, Y indicates data types that are supported on the SAS Workspace Server.

Data Type	CAS Actions	CASL	Data Connectors	Procedures and DATA Step	DS2	FedSQL	Workspace Server Processing
BIGINT			Y		Y		
BLOB		Y					
BOOLEAN		Y	Y				
CHARACTER (CHAR)	Y	Y	Y	Y	Y	Y	Y
DATE		Y	Y		Y		
DATETIME		Y	Y				
DOUBLE	Y	Y	Y	Y	Y	Y	
FLOAT			Y		Y		
IMAGE	Y						
INTEGER			Y		Y		
INT32		Y	Y			Y	
INT64		Y	Y			Y	
ITEMS		Y					
LISTS		Y					
NCHAR			Y		Y		
NUMERIC (NUM)			Y				Y
NVARCHAR			Y		Y		
SMALLINT			Y		Y		
STRING UTF-8		Y					
TABLE		Y					
TIME		Y	Y		Y		
TIMESTAMP			Y		Y		
TINYINT			Y		Y		
VARCHAR	Y	Y	Y	Y	Y	Y	

Additional data types are supported by the data connectors. These data types are first converted to data types that can be processed on the CAS server. Check the data connector documentation for your data source to ensure that a data type is supported.

Note: For more information, see "Data Connectors" in *SAS® Cloud Analytic Services: User's Guide*.

1.2 Essentials of Supervised Prediction



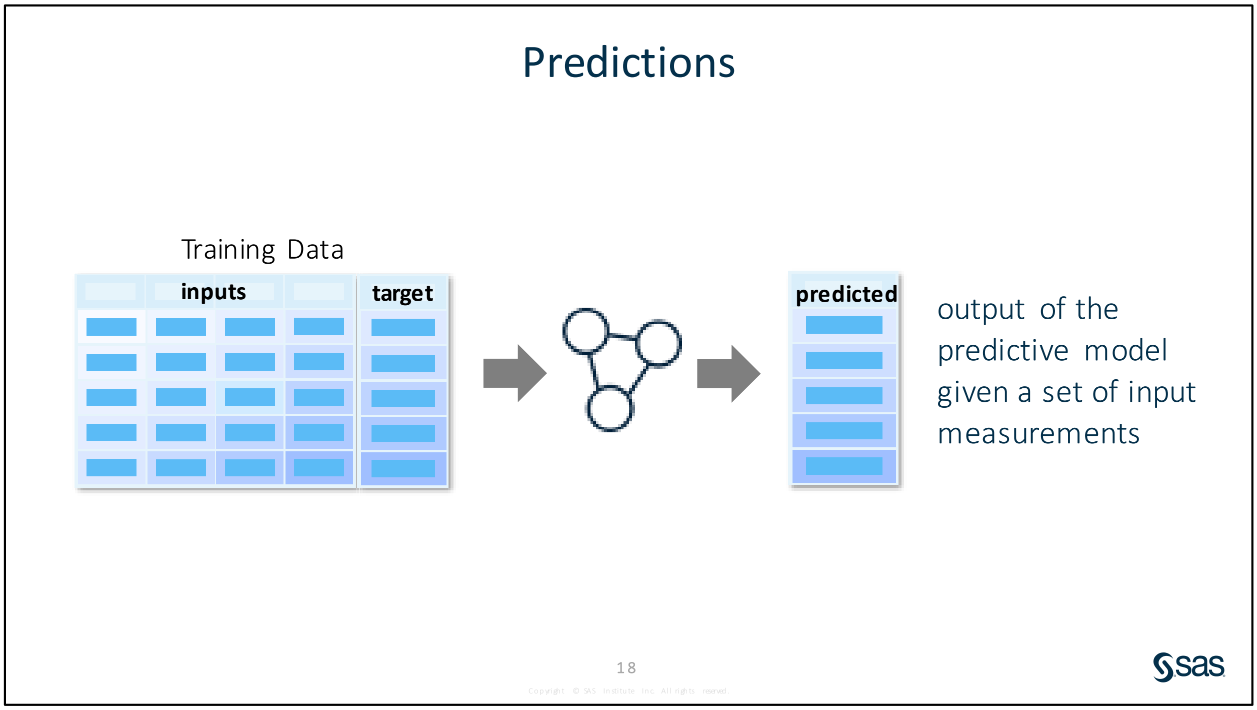
Predictive Model

a concise representation of the input and target association

Predictive modeling (also known as *supervised prediction* or *supervised learning*) starts with a training data set. The observations in a training data set are known as *training cases* (also known as *examples*, *instances*, or *records*). The variables are called *inputs* (also known as *predictors*, *features*, *explanatory variables*, or *independent variables*) and *targets* (also known as a *response*, *outcome*, or *dependent variable*). For a given case, the inputs reflect your state of knowledge before measuring the target.

The measurement scale of the inputs and the target can be varied. The inputs and the target can be numeric variables, such as **income**. They can be nominal variables, such as **occupation**. They are often binary variables, such as a positive or negative response concerning home ownership.

The purpose of the training data is to generate a predictive model. The *predictive model* is a concise representation of the association between the inputs and the target variables.

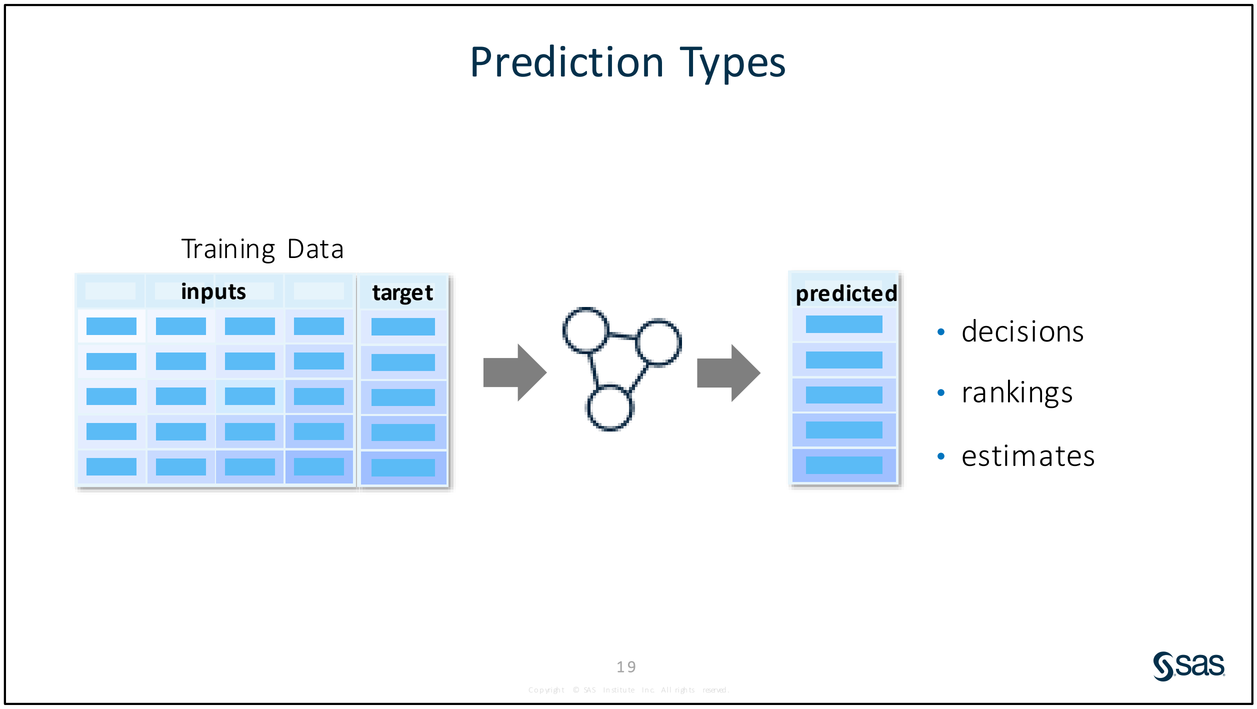


Predictions

output of the predictive model given a set of input measurements

18

The outputs of the predictive model are referred to as *predictions*. Predictions represent your best guess for the target given a set of input measurements. The predictions are based on the associations ***learned*** from the training data by the predictive model.



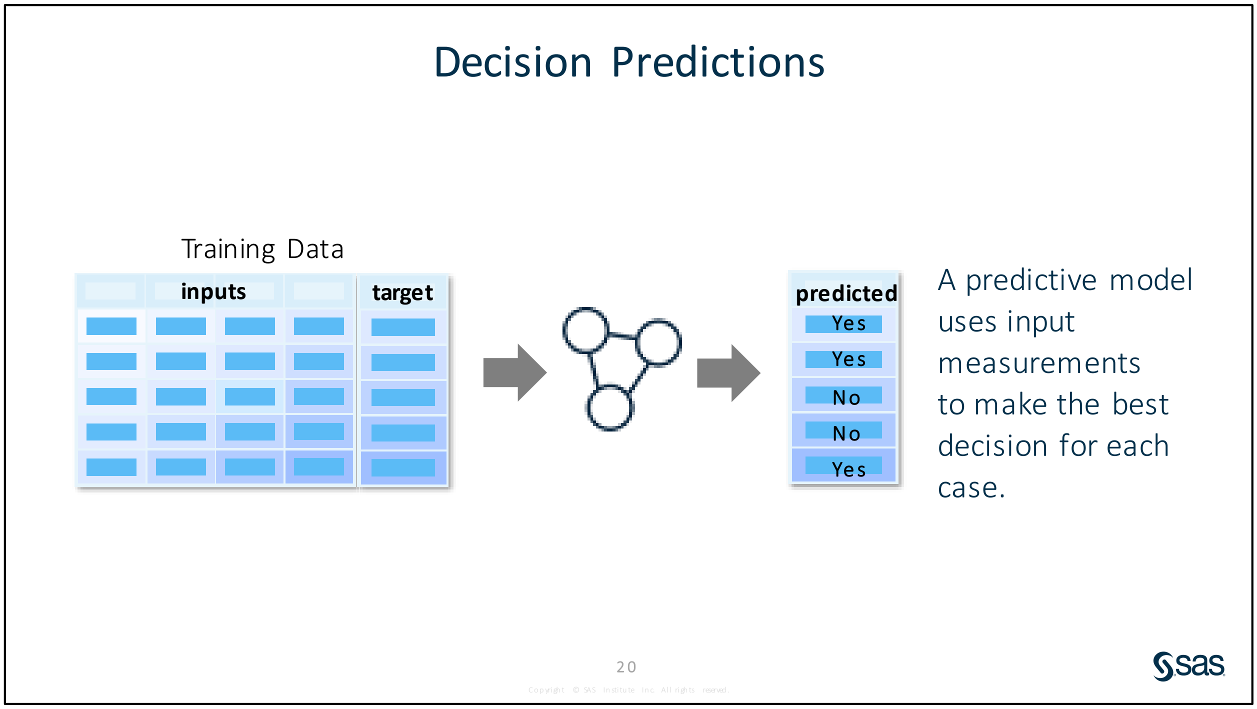
Prediction Types

- decisions
- rankings
- estimates

19

The training data is used to construct a model (rule) that relates the inputs to the target. The predictions can be categorized into three distinct types:

- decisions
- rankings
- estimates.



Decision Predictions

A predictive model uses input measurements to make the best decision for each case.

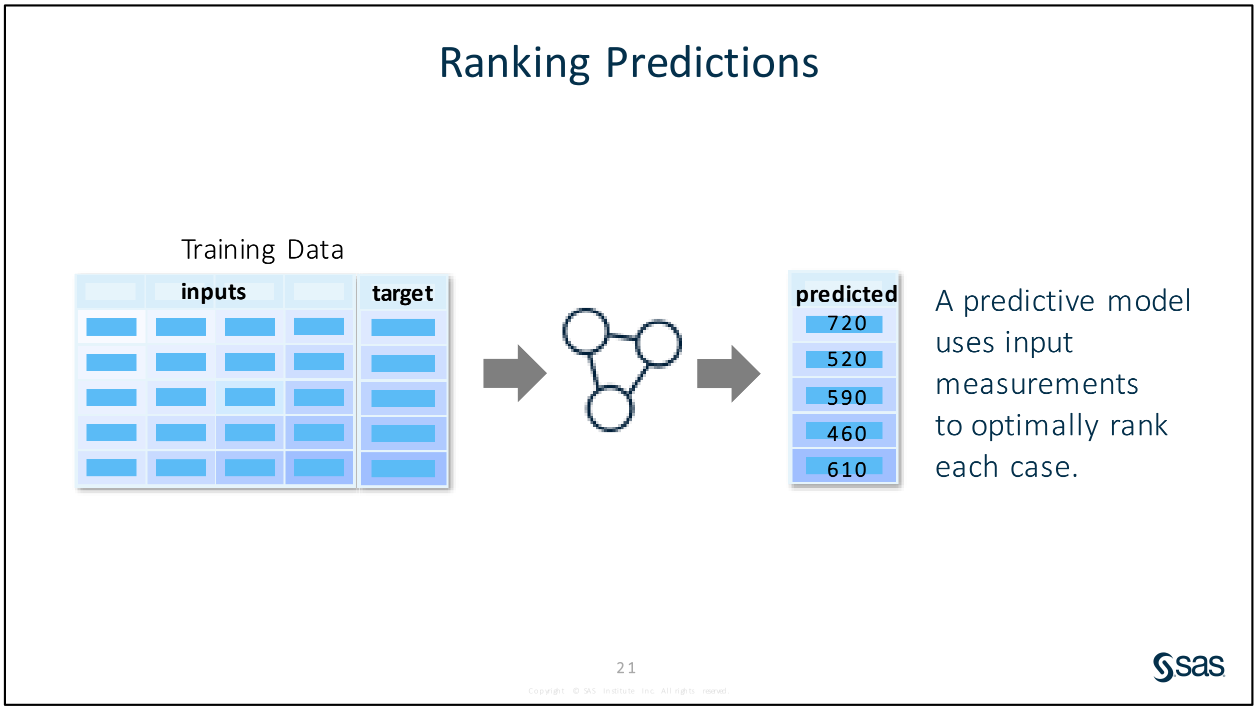
20

Copyright © SAS Institute Inc. All rights reserved.

Decision predictions are the simplest type of prediction. Decisions usually are associated with some type of action (such as classifying a case as a churn or a no-churn). For this reason, decisions are also known as *classifications*. Decision prediction examples include handwriting recognition, fraud detection, and direct mail solicitation.

Decision predictions usually relate to a categorical target variable. For this reason, they are identified as primary, secondary, and tertiary in correspondence with the levels of the target.

Note: Model assessment in Model Studio generally assumes decision predictions when the target variable has a categorical measurement level (binary, nominal, or ordinal).



Ranking Predictions

Training Data

inputs				target

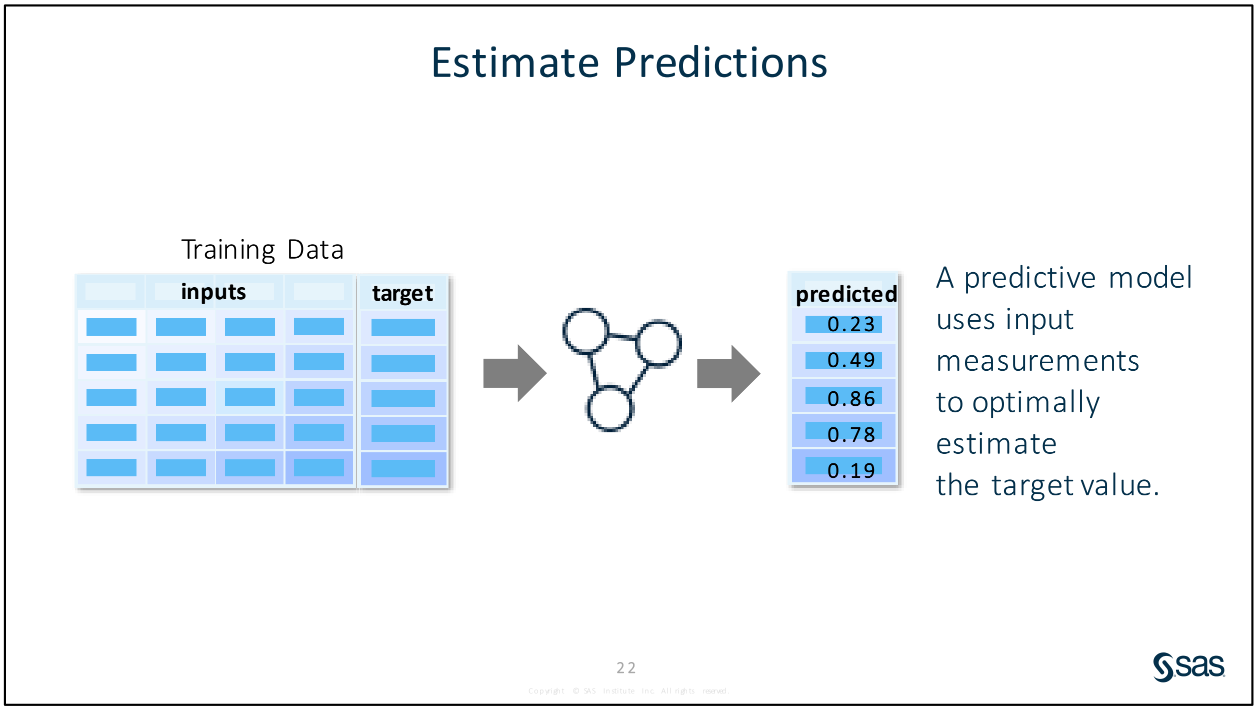
predicted
720
520
590
460
610

A predictive model uses input measurements to optimally rank each case.

21

Ranking predictions order cases based on the input variables' relationships with the target variable. Using the training data, the prediction model attempts to rank ***high value*** cases higher than ***low value*** cases. It is assumed that a similar pattern exists in the scoring data so that ***high value*** cases have high scores. The actual, produced scores are inconsequential; only the relative order is important. The most common example of a ranking prediction is a credit score.

Note: Ranking predictions can be transformed into decision predictions by taking the primary decision for cases above a certain threshold while making secondary and tertiary decisions for cases below the correspondingly lower thresholds. In credit scoring, cases with a credit score above 700 can be called good risks; those with a score between 600 and 700 can be intermediate risks; and those below 600 can be considered poor risks.



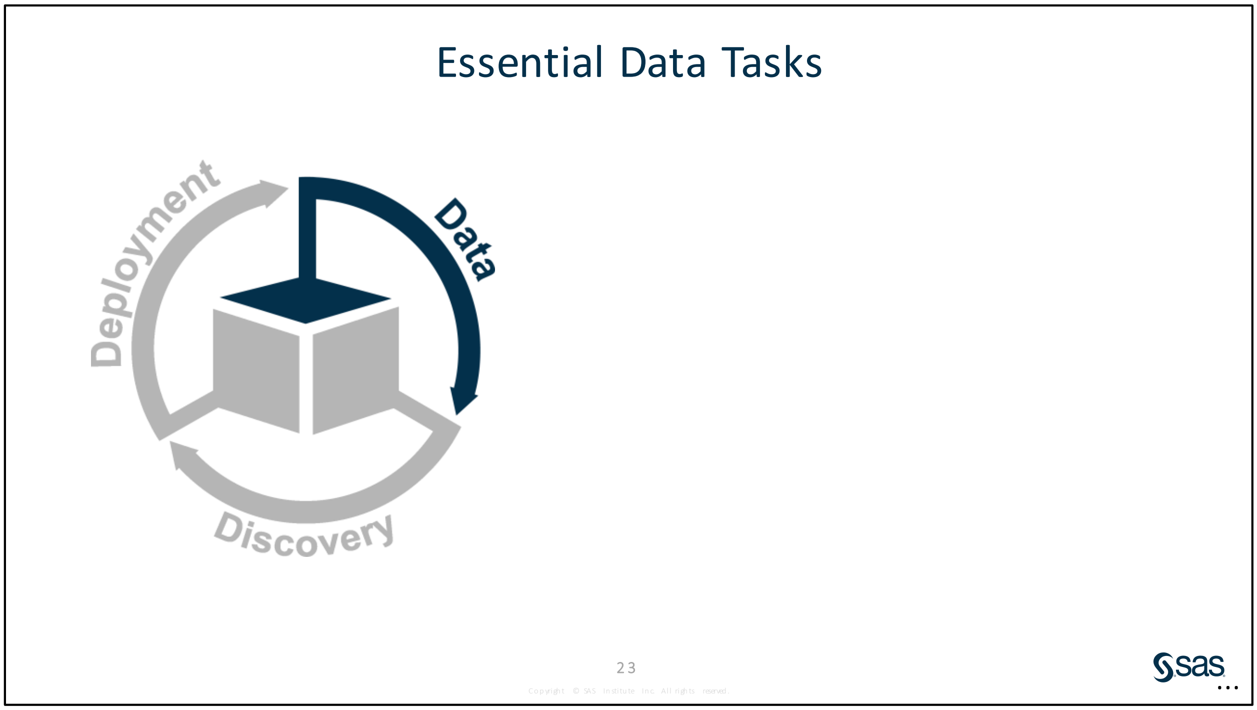
Estimate Predictions

A predictive model uses input measurements to optimally estimate the target value.

Estimate predictions approximate the expected value of the target, conditioned on the input values. For cases with numeric targets, this number can be thought of as the average value of the target for all cases having the observed input measurements. For cases with categorical targets, this number might equal the probability of a target outcome.

Prediction estimates are most commonly used when their values are integrated into a mathematical expression. An example is two-stage modeling, where the probability of an event is combined with an estimate of profit or loss to form an estimate of unconditional expected profit or loss. Prediction estimates are also useful when you are not certain of the ultimate application of the model.

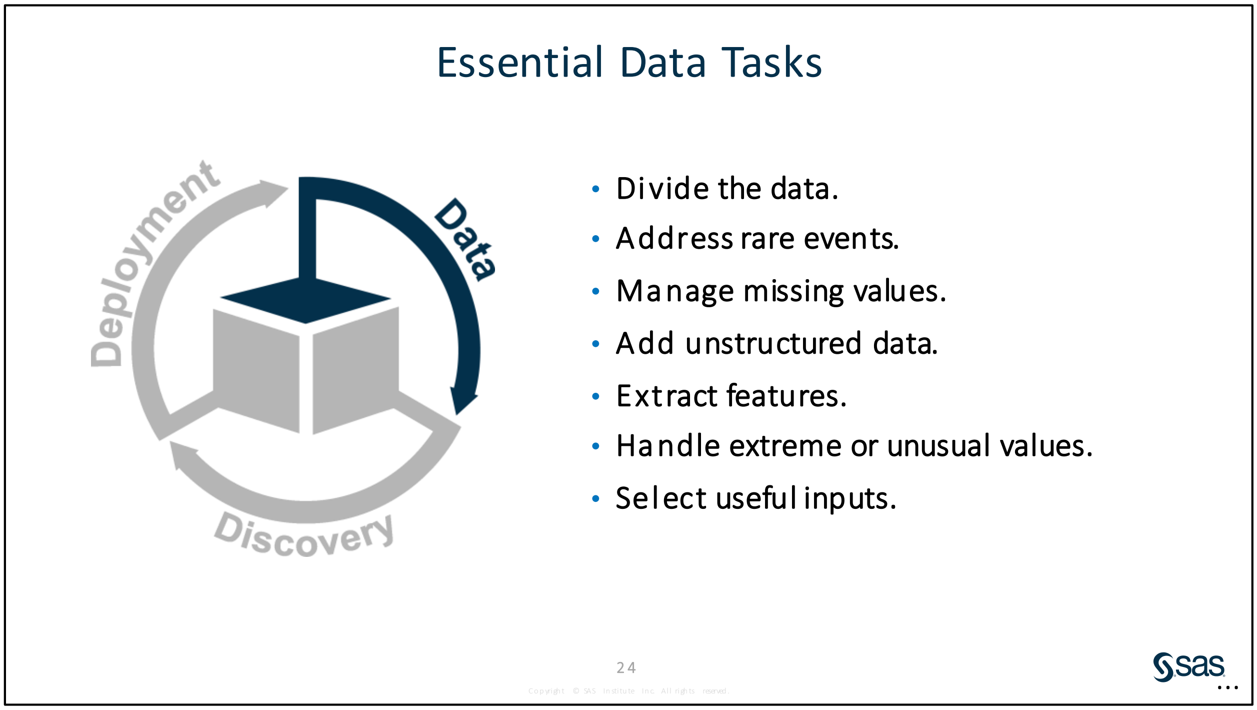
Note: Estimate predictions can be transformed into both decision and ranking predictions. When in doubt, use this option. Most Model Studio modeling tools can be configured to produce estimate predictions.



Essential Data Tasks

23

Effective machine learning models are built on a foundation of well-prepared data. It is commonly proclaimed that 80% of the time spent in devising a successful machine learning application is spent in data preparation (Dasu and Johnson 2003). Data preparation is not strictly about correctly transforming and cleaning existing data. It also includes a good understanding of the features that need to be considered and ensuring that the data at hand are appropriate in the first place. Shortcuts in data preparation will shortchange your models. As they say, “garbage in, garbage out.” Take the time to cultivate your data and be wary of the common challenges described next.



Essential Data Tasks

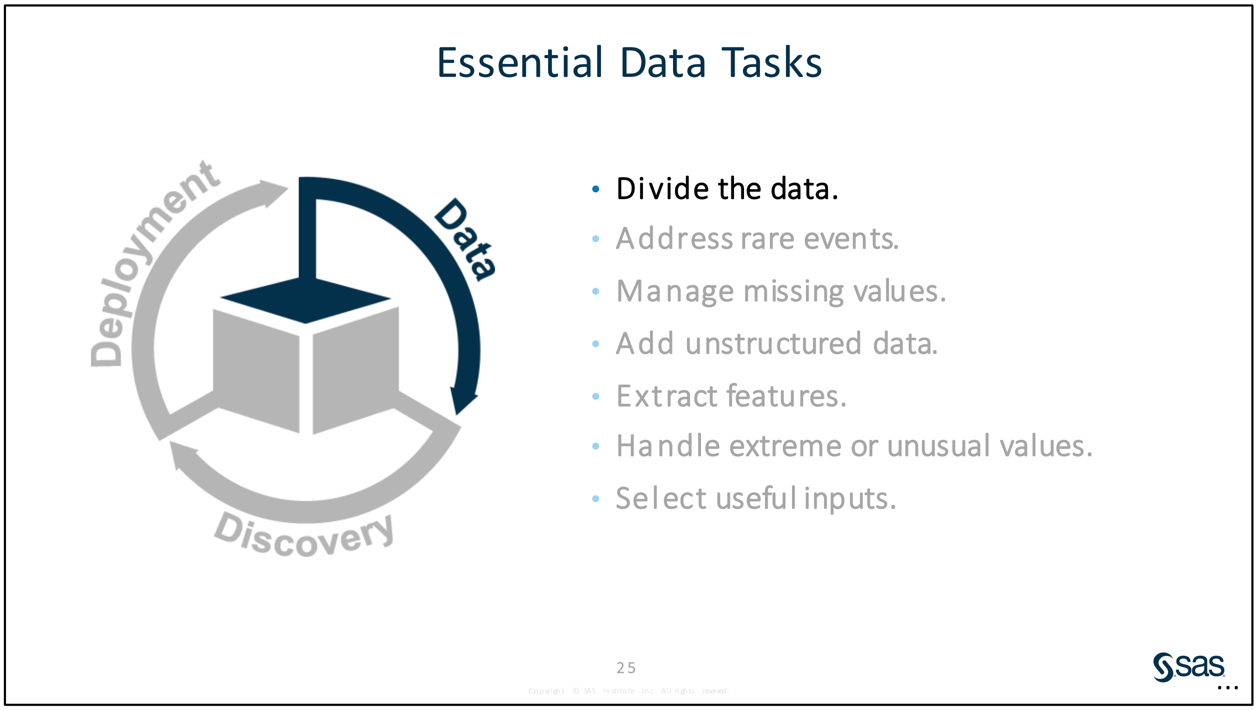
- Divide the data.
- Address rare events.
- Manage missing values.
- Add unstructured data.
- Extract features.
- Handle extreme or unusual values.
- Select useful inputs.

24

Although the predictive models would seem to be the final word in scoring a new case, especially with parametric models like regression and neural network, there are several additional issues that must be addressed.

- *Divide the data:* What should be done for honest assessment of model performance in predictive modeling? How do you tune your model to improve its generalization? You might be tempted to simply avoid the standard strategy of *data splitting* in which a portion is used for fitting the model and the remaining data is separated for empirical validation.
- *Address rare events:* What should be done if you have proportion of events (desired outcome) very low? You might be drawn to make some treatment to make the model robust so that enough events would be used to train the model. Oversampling is one of the treatment to deal rare-event problem.
- *Manage missing values:* What should be done when one of the input values used in the prediction formula is missing? You might be tempted to simply treat the missing value as zero and skip the term involving the missing value. Although this approach can generate a prediction, this prediction is usually biased beyond reason.
- *Add unstructured data:* How should you deal with the overwhelming proliferation of textual data in business? While the amount of textual data is increasing rapidly, businesses' ability to summarize, understand, and make sense of such data for making better business decisions remain challenging.
- *Extract features:* How do you transform the existing features into a lower-dimensional space and generate new features that would be composites of the existing ones? What are techniques that you would use to reduce dimensionality through such a transformation?
- *Handle extreme or unusual values:* How do you score cases with unusual values? Many machine learning algorithms are unaffected by this. However, some are highly sensitive to the problem of outliers. For example, regression models make their best predictions for cases near the centers of the input distributions. If an input can have (on rare occasion) extreme or *outlying* values, the regression should respond appropriately.
- *Select useful inputs:* What would be the optimal set of inputs for your predictive model? Should you simply try every combination of inputs? Unfortunately, the number of models to consider using this approach increases exponentially in the number of available inputs. Such an exhaustive search is impractical for realistic prediction problems.

The above issues affect both model construction (the discovery phase) and model scoring (the deployment phase). The first of these, model complexity and event-based sampling, are dealt with immediately. The remaining issues are addressed subsequently in this chapter and the next one.

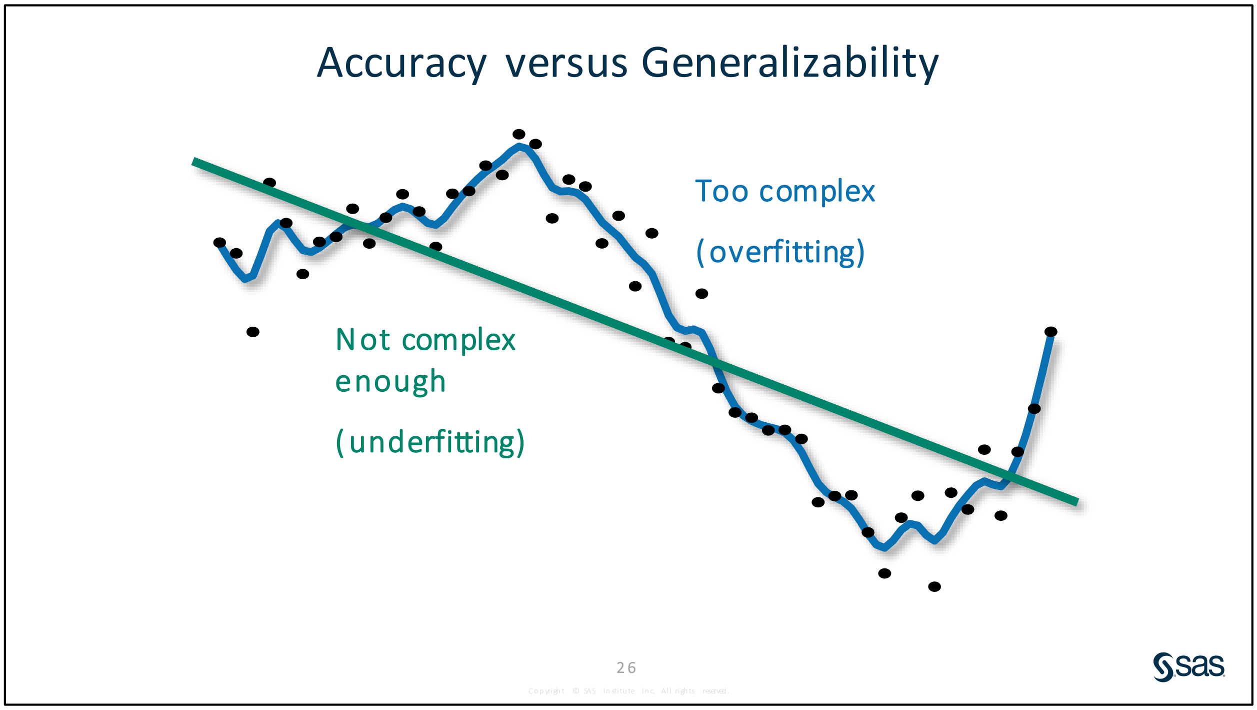


Essential Data Tasks

- Divide the data.
- Address rare events.
- Manage missing values.
- Add unstructured data.
- Extract features.
- Handle extreme or unusual values.
- Select useful inputs.

25

In typical machine learning tasks, data are divided into different sets (partitions): some data for training the model and some data for evaluating the model.

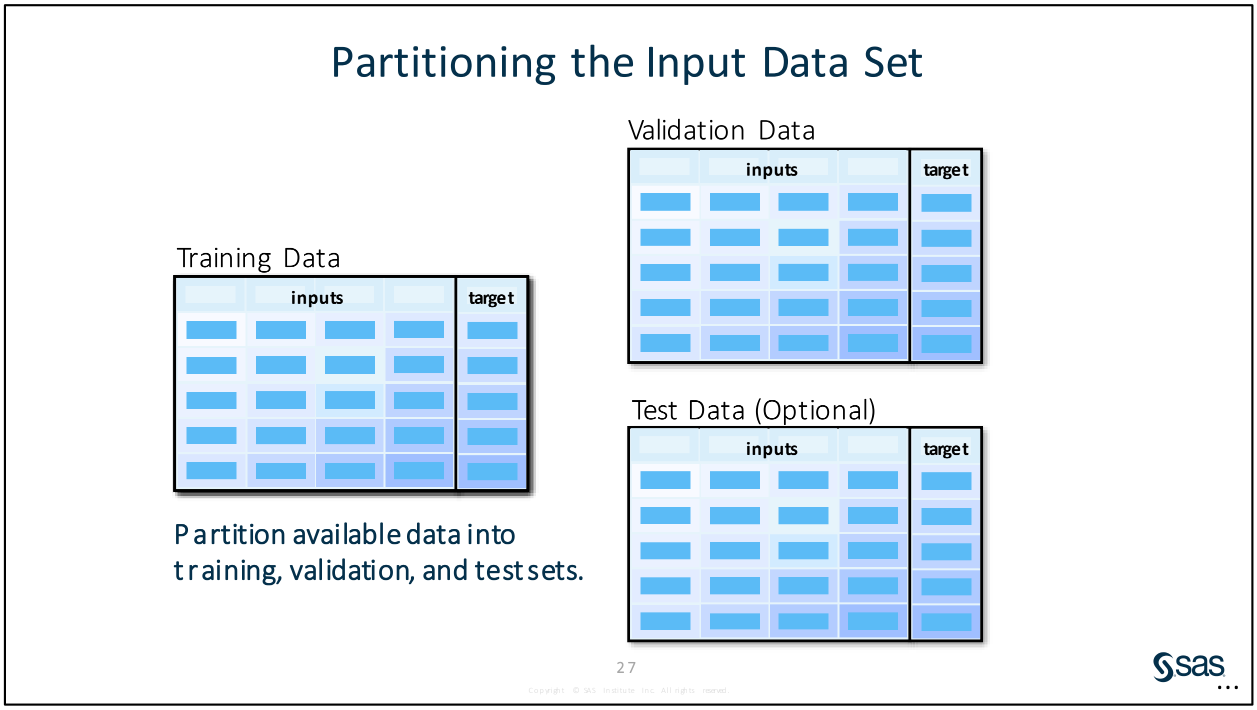


Accuracy versus Generalizability

26

Fitting a model to data requires searching through the space of possible models. Constructing a model with good generalization requires choosing the right complexity. Selecting model complexity involves a trade-off between bias and variance. An insufficiently complex model might not be flexible enough, which can lead to *underfitting*—that is, systematically missing the signal (high bias).

A naïve modeler might assume that the most complex model should always outperform the others, but this is not the case. An overly complex model might be too flexible, which can lead to *overfitting*—that is, accommodating nuances of the random noise in the sample (high variance). A model with the right amount of flexibility gives the best generalization.



Partitioning the Input Data Set

Validation Data

inputs | target

Training Data

inputs | target

Test Data (Optional)

inputs | target

Partition available data into training, validation, and test sets.

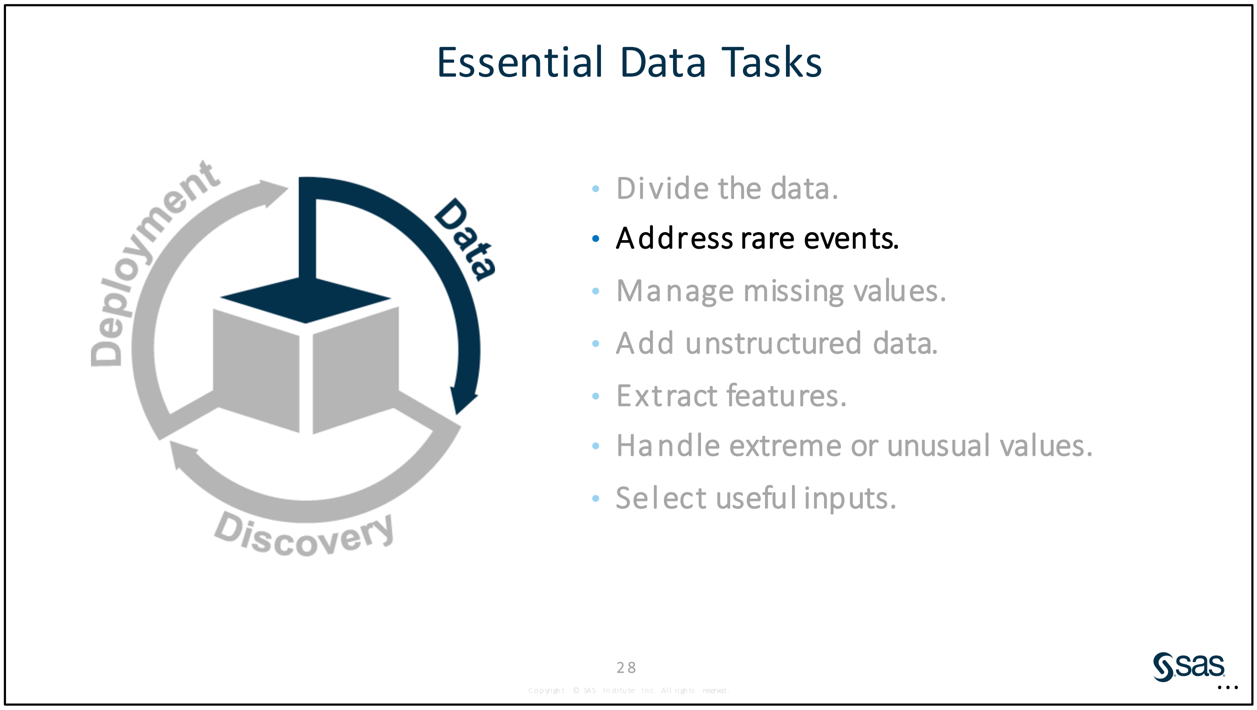
27

The data are split into at least two, but not more than three, non-overlapping groups. The first partition of the data, the *training set*, is used to build models.

Usually, for each modeling algorithm, a series of models is constructed, and the models increase in their complexity. The idea behind constructing a series of models is that some will be too simple (underfit) and others will be too complex (overfit). Each of the models is assessed for its performance on the second partition of the data, the *validation set*. In this way, the validation data is used to “optimize complexity” of the model and find the sweet spot between being underfit and being overfit. Validation data is used to fine tune the models built on training data and determines whether additional training is required.

The *test data set* is an optional partition for the model building process, but some industries might require it as a source of unbiased model performance. The test data gives the honest unbiased estimates of the model. The test data set provides one final measure of how the model performs on new data, before the model is put into production. It assesses only the final model’s performance on unused data. However, it might also be used to select the best model by scoring the champion model based on the training and the validation data. Model Studio by default uses the validation data for selecting the champion model.

It is critical that all transformations that are used to prepare the training data are applied to any validation, test, or other holdout data. In other words, it is critical that information from test data or holdout data does not leak into the training data. Information leakage can occur in many ways and can potentially lead to overfitting or overly optimistic error measurements. For example, think of taking a mean or median across your data before partitioning and then later using this mean or median to impute missing values across all partitions of your data. In this case, your training data would be aware of information from your validation, test, or holdout partitions. To avoid this type of leakage, values for imputation and other basic transformations should be generated from only the training data or within each partition independently.

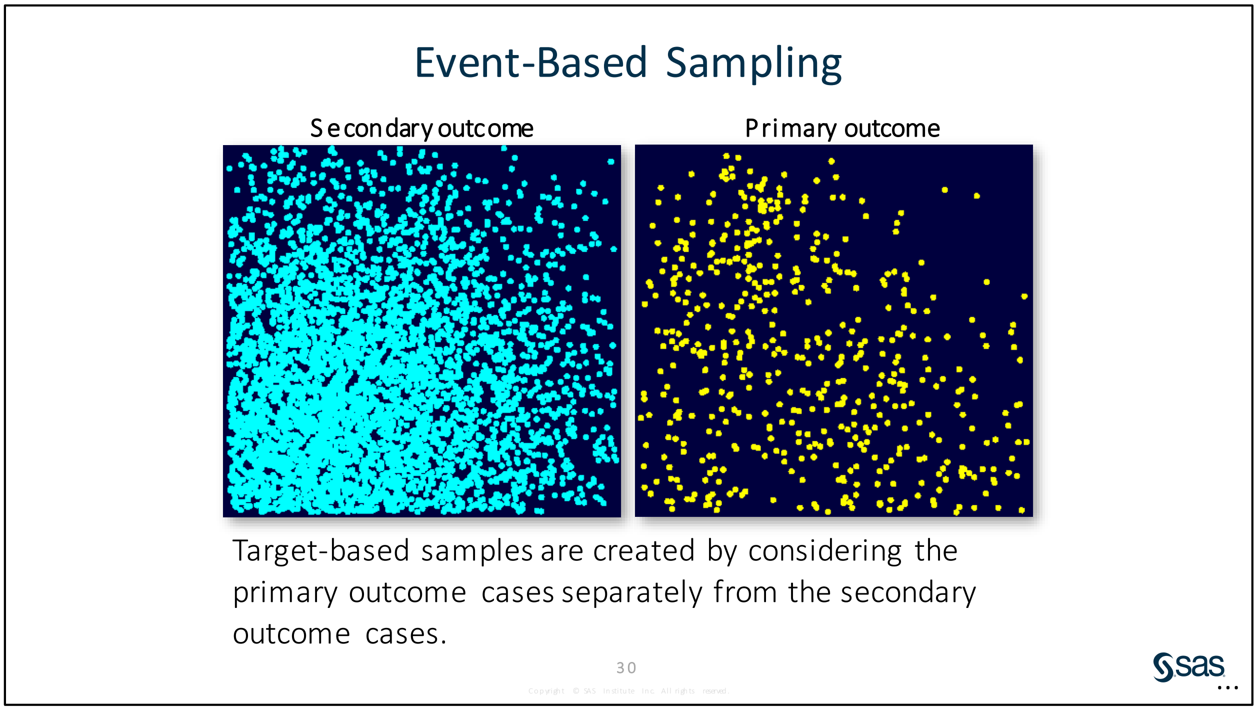


Essential Data Tasks

- Divide the data.
- Address rare events.
- Manage missing values.
- Add unstructured data.
- Extract features.
- Handle extreme or unusual values.
- Select useful inputs.

28

It is also important to be mindful of your target of interest and understand whether it can be characterized as a rare event relative to your total number of samples. Applications such as detecting fraudulent activity must take special steps to ensure that the data used to train the model include a representative number of fraudulent samples in to capture the event sufficiently. (For example, 1 out of every 1,000 credit card transactions is fraudulent.) Fitting a model to such data without accounting for the extreme imbalance in the occurrence of the event gives you a model that is extremely accurate at telling you absolutely nothing of value. Special sampling methods that modify an imbalanced data set are commonly used to provide a more balanced distribution when modeling rare events (He and Garcia 2009).



Event-Based Sampling

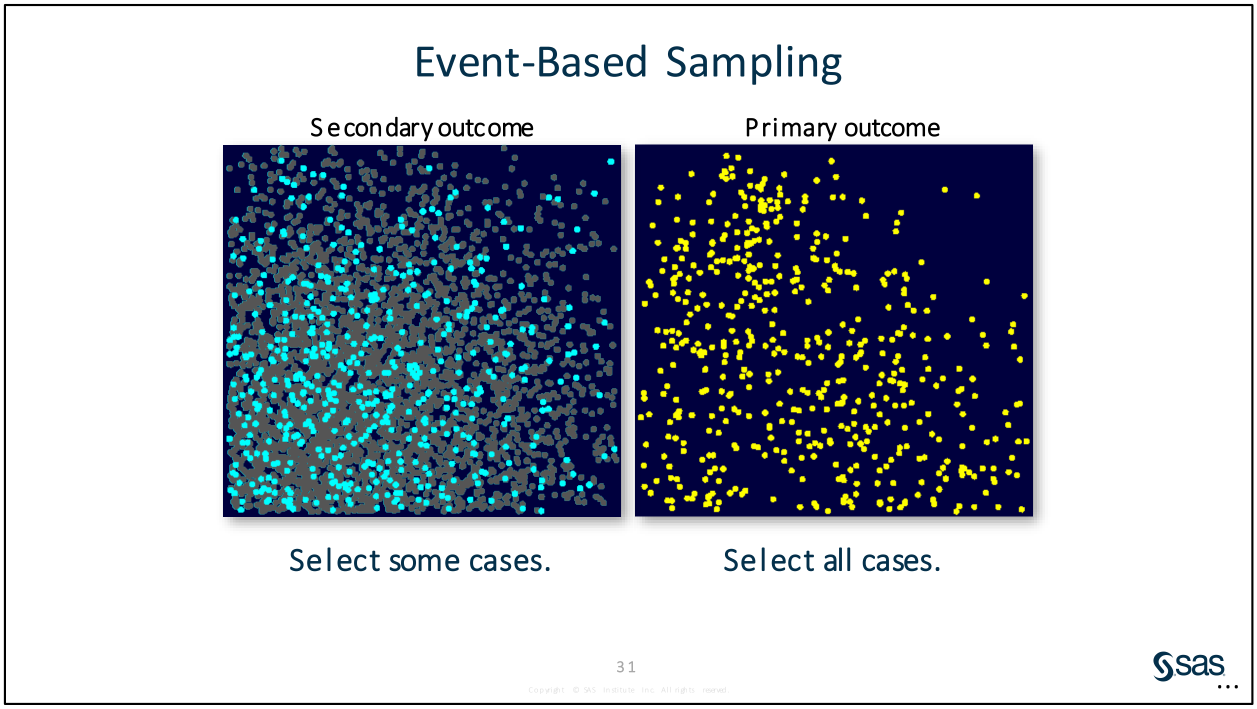
Secondary outcome

Primary outcome

Target-based samples are created by considering the primary outcome cases separately from the secondary outcome cases.

30

A common predictive modeling practice is to build models from a sample with a primary outcome proportion different from the true population proportion. This is typically done when the ratio of primary to secondary outcome cases is small.



Event-Based Sampling

Secondary outcome

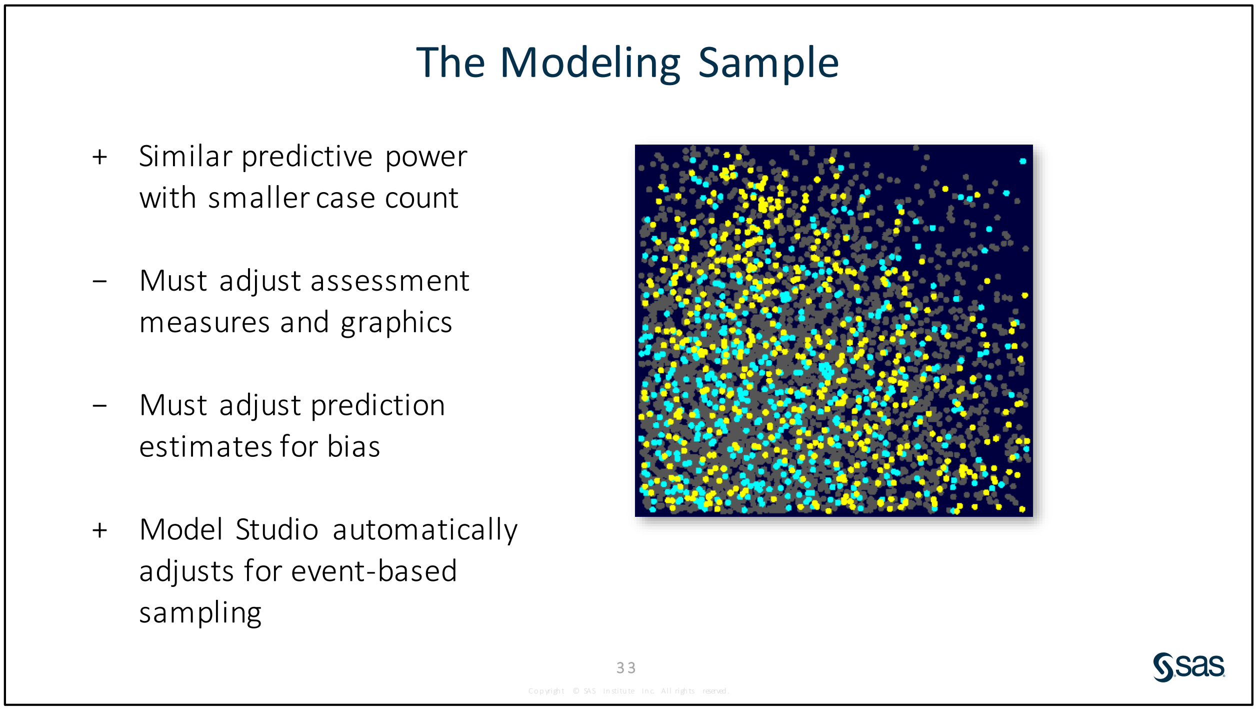
Primary outcome

Select some cases.

Select all cases.

31

Event-based sampling derives its name from the technique used to generate the modeling data. That is, samples are drawn separately based on the target events and non-events. In the case of a rare event, usually all events are selected. Then each event outcome is matched by one or (optimally) more non-event outcomes.



The Modeling Sample

+ Similar predictive power with smaller case count

– Must adjust assessment measures and graphics

– Must adjust prediction estimates for bias

+ Model Studio automatically adjusts for event-based sampling

33

The advantage of event-based sampling is that you can obtain (on the average) a model of similar predictive power with a smaller overall case count. This is in concordance with the idea that the amount of information in a data set with a categorical outcome is determined not by the total number of cases in the data set itself, but instead by the number of cases in the rarest outcome category. For binary target data sets, this is usually the event outcome. (Harrell 2006)

This advantage might seem of minimal importance in the age of extremely fast computers. However, the model-fitting process occurs only after the completion of a long, tedious, and error-prone data preprocessing. Smaller sample sizes for data preprocessing are usually welcome.

Although it reduces analysis time, event-based sampling also introduces some analysis complications:

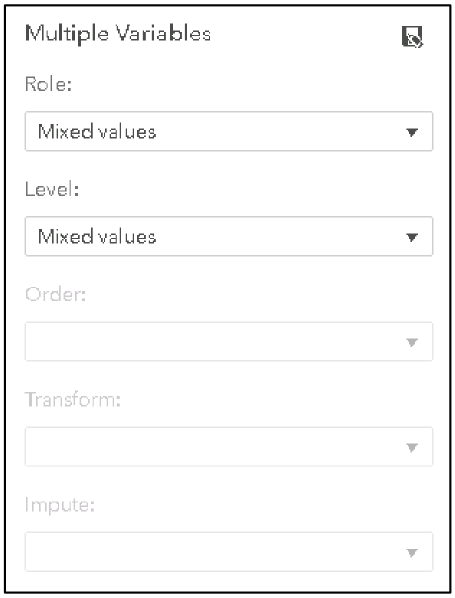
- Most model fit statistics (especially those related to prediction decisions) and most of the assessment plots are closely tied to the outcome proportions in the training samples. If the outcome proportion in the training and validation samples do not match the outcome proportions in the scoring population, model performance can be greatly misestimated.
- If the outcome proportions in the training sample and scoring populations do not match, model prediction estimates are biased.

Note: Model Studio automatically adjusts assessment measures, assessment graphs, and prediction estimates for bias. After running the pipeline, which executes an automated sequence of steps to build models, you can examine the score code. The score code contains a section titled **Adjust Posterior Probabilities**. This code block modifies the posterior probability by multiplying it by the ratio of the actual probability to the event-based sampling values specified previously.

Modifying the Data Partition

In this demonstration, you modify metadata roles of some variables, explore the advanced project settings, and change the data partition properties.

1. Ensure that the **Demo** project is open and that **churn** is not selected. Reopen the project if you have closed it and deselect **churn** if it is selected by clicking the check box next to the variable's name.
2. Modify the following properties of the variables specified:
   a. To specify properties of a variable on the Data tab, first select the desired variable. (You can select the check boxes of several variables at one time.)
   b. Next, in the right pane, select the new role or level of the variables.



Multiple Variables

Role: Mixed values

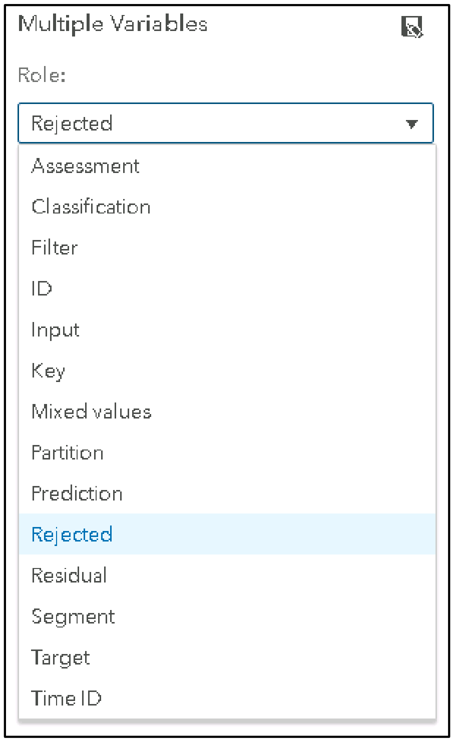
Level: Mixed values

Note: Variable metadata includes the role and measurement level of the variable. Common variable roles are Input, Target, Rejected, Text, and ID. Common variable measurement levels are Interval, Binary, Nominal, and Ordinal. See the appropriate drop-down menus in Model Studio for the full list of variable roles and measurement levels.

For the following variables, change the role to **Rejected**:

- **city**
- **city_lat**
- **city_long**
- **data_usage_amt**
- **mou_onnet_6m_normal**
- **mou_roam_6m_normal**
- **region_lat**
- **region_long**
- **state_lat**

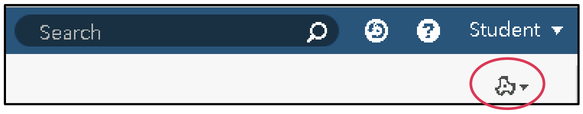
- **state_long**
- **tweedie_adjusted**


Multiple Variables

Role:

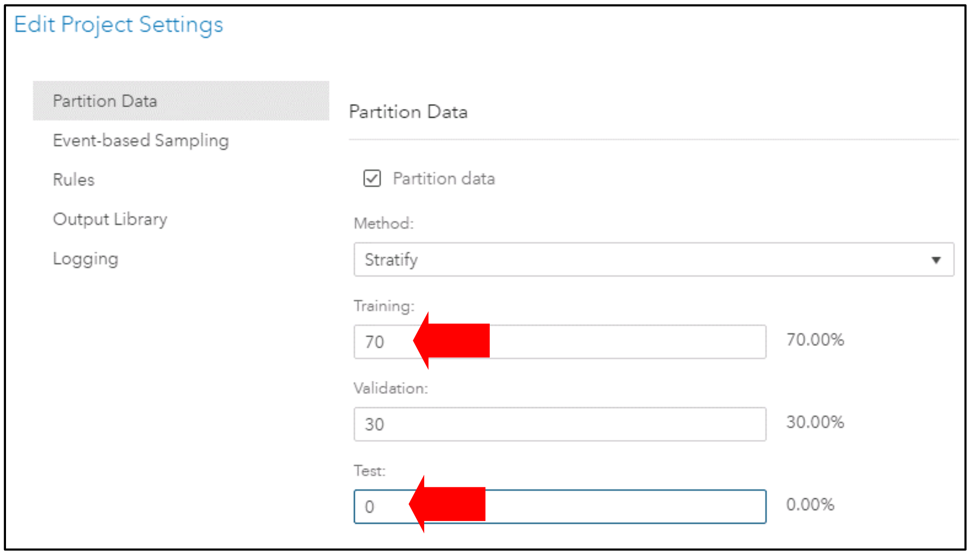
Rejected

3. Change the default partition by clicking ⚙ (**Settings**) in the upper right corner of the window.



Note: When you created a new project in Model Studio, by default, partitioning was performed. If you want to see or modify the partition settings before creating the project, you can do this from the user settings. In the user settings, the Partition tab enables you to specify the method for partitioning as well as associated percentages. Any settings at this level are global and will be applied to any new project created.

4. Select **Project settings**. With **Partition Data** selected in the Edit Project Settings window, change the Training percentage to 70 and the Test percentage to **0**.



Edit Project Settings

Partition Data

☑ Partition data

Method: Stratify

Training: 70 70.00%

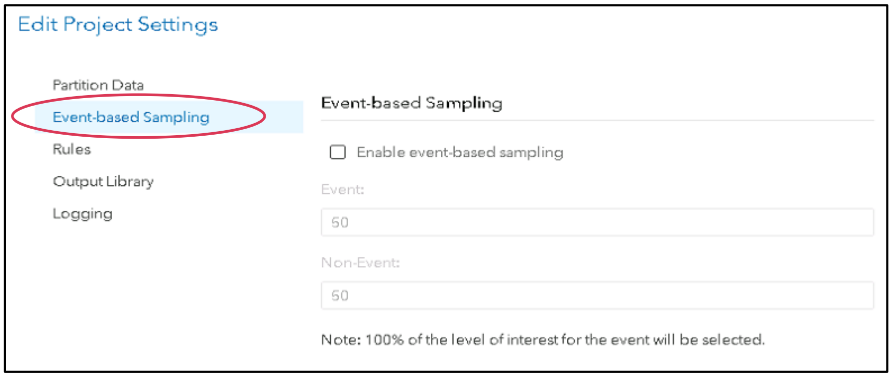
Validation: 30 30.00%

Test: 0 0.00%

These settings can be edited only if no pipelines in the project have been run. After the first pipeline has been run, the partition tables are created for the project and partition settings cannot be changed.

Note: Recall that it was shown in the last demonstration that the Partition Data options could also be accessed and changed while the project is being created, under the Advanced settings.

5. Select **Event-based Sampling**.



Edit Project Settings

Event-based Sampling

☐ Enable event-based sampling

Event: 50

Non-Event: 50

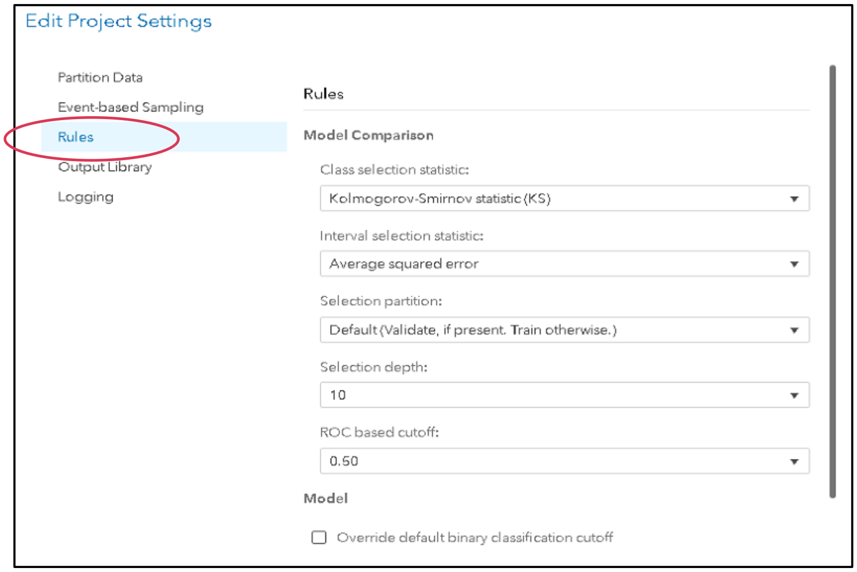
Note: 100% of the level of interest for the event will be selected.

When event-based sampling is turned on (it is off by default), the desired proportion of event and non-event cases can be set after the sampling is done. When turned on, the default proportions for both events and non-events after sampling is 50% for each. The sum of both must be 100%. ***After a pipeline has been run in the project, the Event-based Sampling settings cannot be changed.***

Note: Recall that it was shown in the last demonstration that the Event-based Sampling options could also be accessed and changed while the project is being created, under the Advanced settings.

Keep the Event-based Sampling options at their default settings.

6. Select **Rules**.



Edit Project Settings

Rules

Model Comparison

Class selection statistic: Kolmogorov-Smirnov statistic (KS)

Interval selection statistic: Average squared error

Selection partition: Default (Validate, if present. Train otherwise.)

Selection depth: 10

ROC based cutoff: 0.50

Model

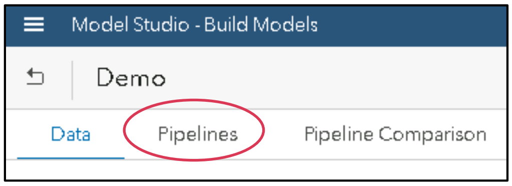
□ Override default binary classification cutoff

The Rules options can be used to change the selection statistic and partitioned data set that determine the champion model during model comparison. Statistics can be selected for class and interval targets.

Keep the Rules options at their default settings.

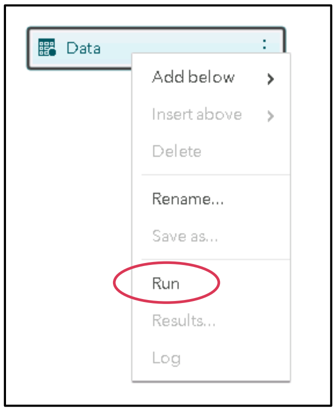
7. Be sure to click **Save** because the partition options were changed.

8. Click the **Pipelines** tab in the Demo project.

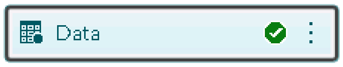


On the Pipelines tab, you can create, modify, and run pipelines. Each pipeline has a unique name and optional description.

9. Right-click the **Data** node and select **Run**.

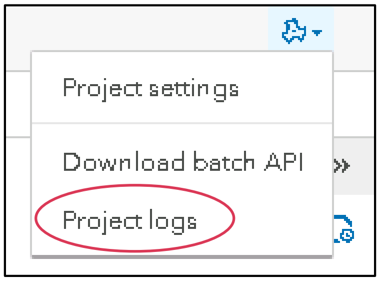


The green check mark in the node indicates that it ran without error. The partition is successfully created.

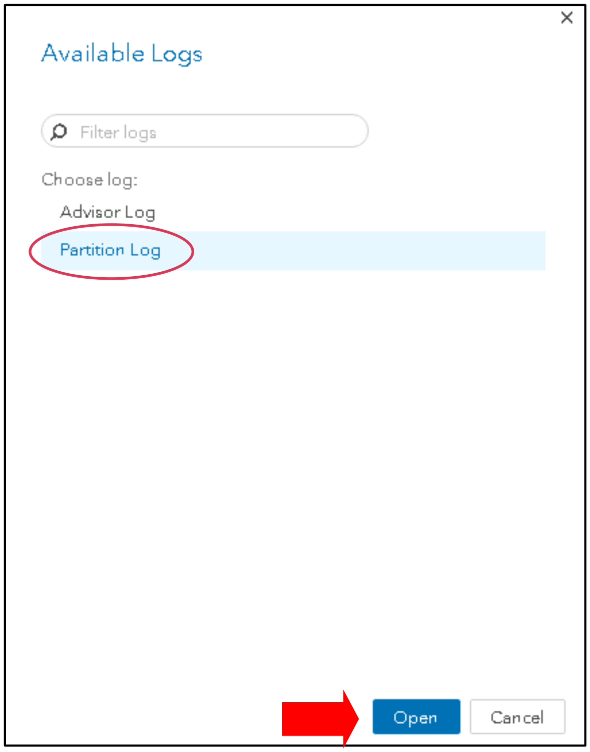


Note: After you run the Data node, you cannot change the partitioning, event-based sampling, project metadata, project properties, or the target variable. However, you can change variable metadata with the Manage Variables node.

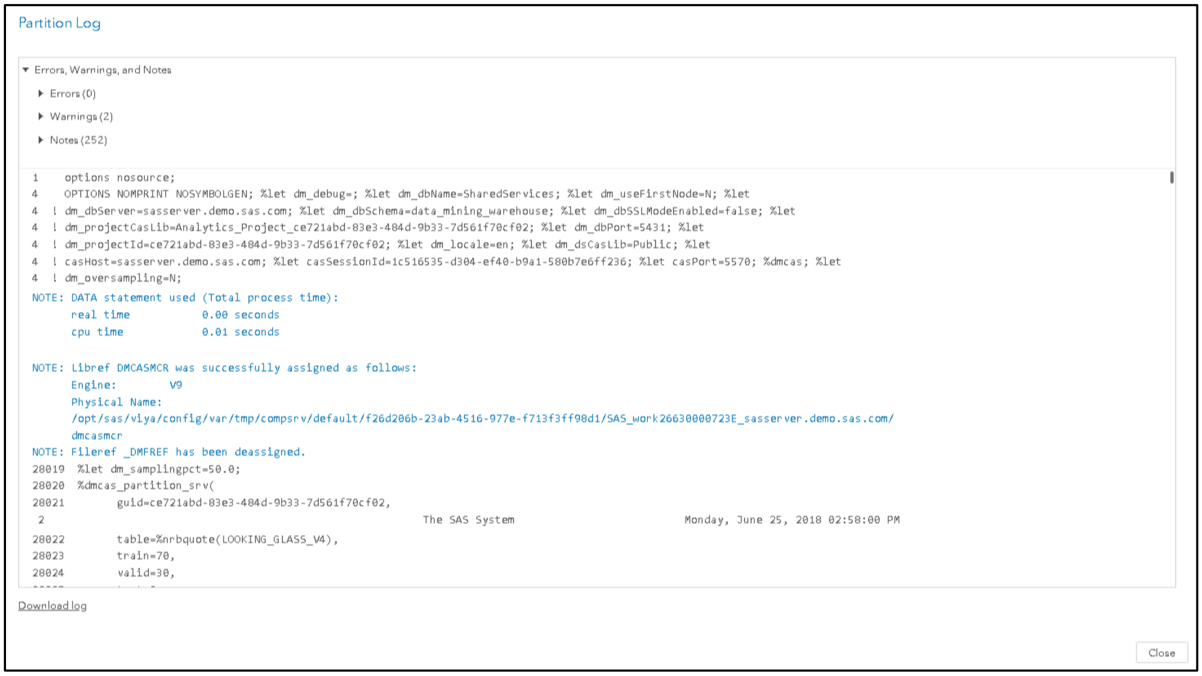
10. The log file for this partitioning action can be viewed. Click **Settings** in the upper right corner.
11. Select **Project logs**.



12. From the Available Logs window, select **Partition Log** and then click **Open**.



The log file can be viewed and even downloaded.



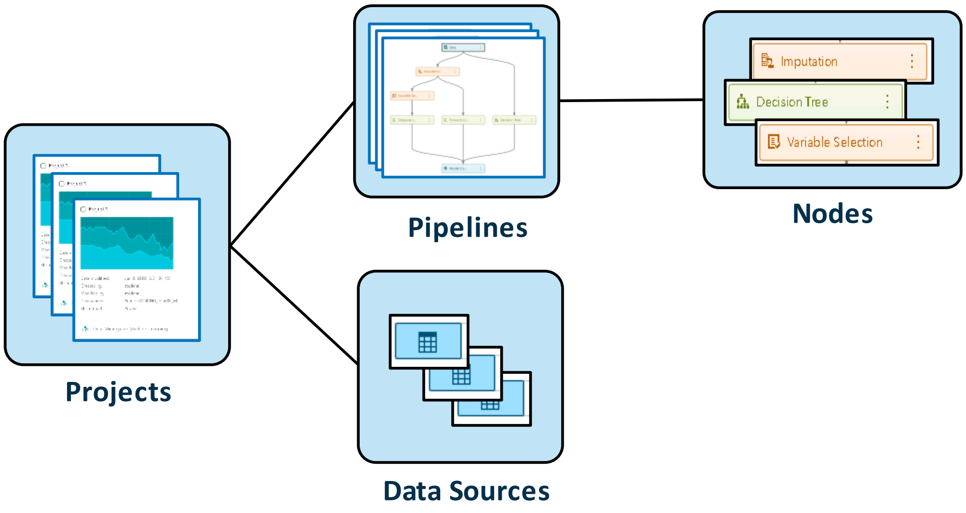
Partition Log

```
1     options nosource;
4     OPTIONS NOMPRINT NOSYMBOLGEN; %let dm_debug=; %let dm_dbName=SharedServices; %let dm_useFirstNode=N; %let
4   ! dm_dbServer=sasserver.demo.sas.com; %let dm_dbSchema=data_mining_warehouse; %let dm_dbSSLModeEnabled=false; %let
4   ! dm_projectCasLib=Analytics_Project_ce721abd-83e3-484d-9b33-7d561f70cf02; %let dm_dbPort=5431; %let
4   ! dm_projectId=ce721abd-83e3-484d-9b33-7d561f70cf02; %let dm_locale=en; %let dm_dsCasLib=Public; %let
4   ! casHost=sasserver.demo.sas.com; %let casSessionId=1c516535-d304-ef40-b9a1-580b7e6ff236; %let casPort=5570; %dmcas; %let
4   ! dm_oversampling=N;
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds

NOTE: Libref DMCASMCR was successfully assigned as follows:
      Engine:        V9
      Physical Name:
      /opt/sas/viya/config/var/tmp/compsrv/default/f26d206b-23ab-4516-977e-f713f3ff98d1/SAS_work26630000723E_sasserver.demo.sas.com/
      dmcasmcr
NOTE: Fileref _DMFREF has been deassigned.
28019  %let dm_samplingpct=50.0;
28020  %dmcas_partition_srv(
28021        guid=ce721abd-83e3-484d-9b33-7d561f70cf02,
 2                                                    The SAS System                          Monday, June 25, 2018 02:58:00 PM

28022        table=%nrbquote(LOOKING_GLASS_V4),
28023        train=70,
28024        valid=30,
```

13. Click **Close** ⇨ **Cancel** to return to the pipeline**.**

End of Demonstration

Analysis Elements in Model Studio



Projects

Pipelines

Nodes

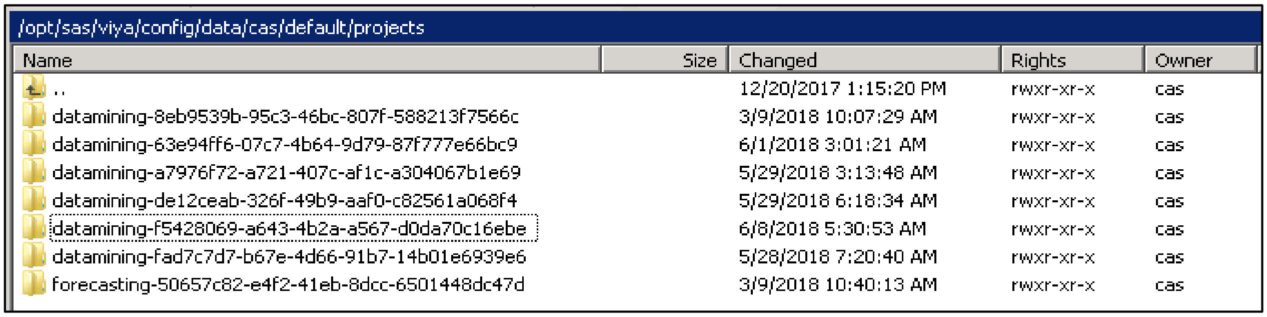
Data Sources

35

A project is a top-level container for your analytic work in Model Studio. A Model Studio project contains the data source, the pipelines that you create, and related project metadata (such as project type, project creator, share list, and last update history). If you create more than one pipeline in your project, analytic results that compare the performance of multiple pipelines are also stored in the project.

In Model Studio, metadata is defined as the set of variable roles, measurement levels, and other configurations that apply to your data set. When creating multiple projects using similar data sets (or when using a single data set), you might find it useful to store the metadata configurations for usage across projects. Model Studio enables you to do this by collecting the variables in a repository named Global Metadata. By storing your metadata configurations as global metadata, they will apply to new data sets that contain variables with the same name.

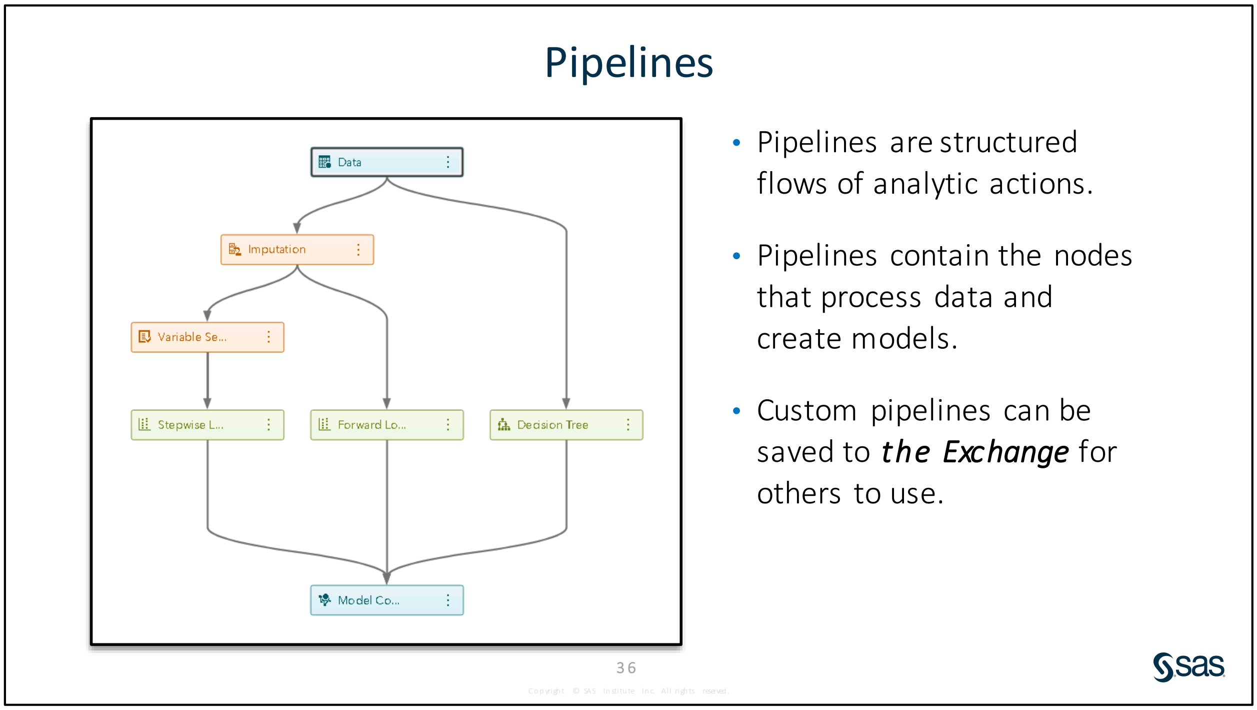
You can add nodes to the pipeline to create your modeling process flow.



/opt/sas/viya/config/data/cas/default/projects

Name	Size	Changed	Rights	Owner
..		12/20/2017 1:15:20 PM	rwxr-xr-x	cas
datamining-8eb9539b-95c3-46bc-807f-588213f7566c		3/9/2018 10:07:29 AM	rwxr-xr-x	cas
datamining-63e94ff6-07c7-4b64-9d79-87f777e66bc9		6/1/2018 3:01:21 AM	rwxr-xr-x	cas
datamining-a7976f72-a721-407c-af1c-a304067b1e69		5/29/2018 3:13:48 AM	rwxr-xr-x	cas
datamining-de12ceab-326f-49b9-aaf0-c82561a068f4		5/29/2018 6:18:34 AM	rwxr-xr-x	cas
datamining-f5428069-a643-4b2a-a567-d0da70c16ebe		6/8/2018 5:30:53 AM	rwxr-xr-x	cas
datamining-fad7c7d7-b67e-4d66-91b7-14b01e6939e6		5/28/2018 7:20:40 AM	rwxr-xr-x	cas
forecasting-50657c82-e4f2-41eb-8dcc-6501448dc47d		3/9/2018 10:40:13 AM	rwxr-xr-x	cas

You can view the list of projects (like above) by navigating to the location where it has been saved. Shown above is the path for a Linux OS using WinSCP (a File Transfer Protocol application on your client machine). Generally, the path is **/opt/sas/viya/config/data/cas/default/projects/**. You might have a different path if it is a Windows installation.



Pipelines

- Pipelines are structured flows of analytic actions.
- Pipelines contain the nodes that process data and create models.
- Custom pipelines can be saved to ***the Exchange*** for others to use.

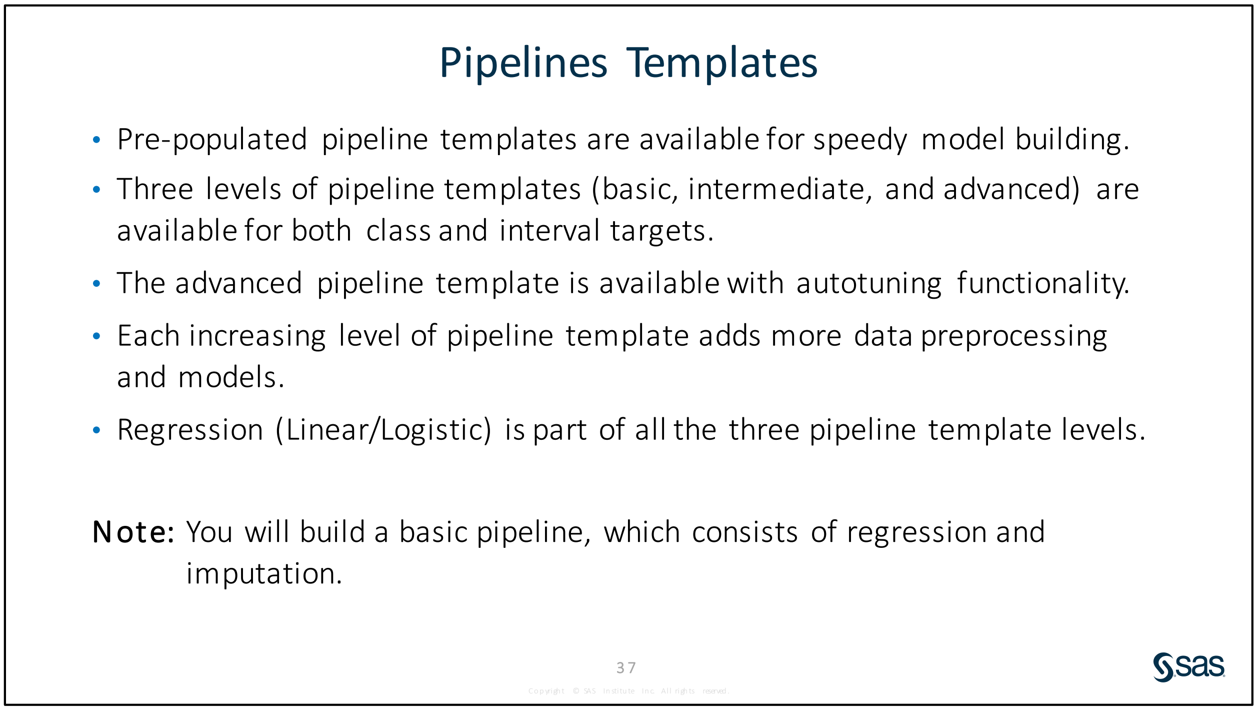
36

In Model Studio, you can create analytic process flow in the form of a pipeline. After creating a new pipeline, you can create visual data mining functionality by adding nodes to the pipeline. Nodes can be added separately or, to save time, templates can add several nodes at once. To create a pipeline from a template, specify the template in the New Pipeline window. You can add nodes to a pipeline in two ways:

1. dragging and dropping from an expanded Nodes pane
2. right-click and select either **Add below** or **Insert above**.

Pipelines are grouped together in a top-level container (that is, in a project that also includes the data set that you want to model and a pipeline comparison tool). A project can contain multiple pipelines. You can create a new pipeline and modify an existing pipeline.

Pipelines can be saved to the Exchange where they become accessible to other users. All available nodes, along with descriptions, and all available pipeline templates, including prebuilt and user created, can be found here.



Pipelines Templates

- Pre-populated pipeline templates are available for speedy model building.
- Three levels of pipeline templates (basic, intermediate, and advanced) are available for both class and interval targets.
- The advanced pipeline template is available with autotuning functionality.
- Each increasing level of pipeline template adds more data preprocessing and models.
- Regression (Linear/Logistic) is part of all the three pipeline template levels.

Note: You will build a basic pipeline, which consists of regression and imputation.

37

Model Studio supports templates as a method for creating statistical models quickly. A template is a special type of pipeline that is pre-populated with configurations that can be used to create a model. A template might consist of multiple nodes or a single node. Model Studio includes a set of templates that represent frequent use cases, but you can also create models themselves and save them as templates in the toolkit.

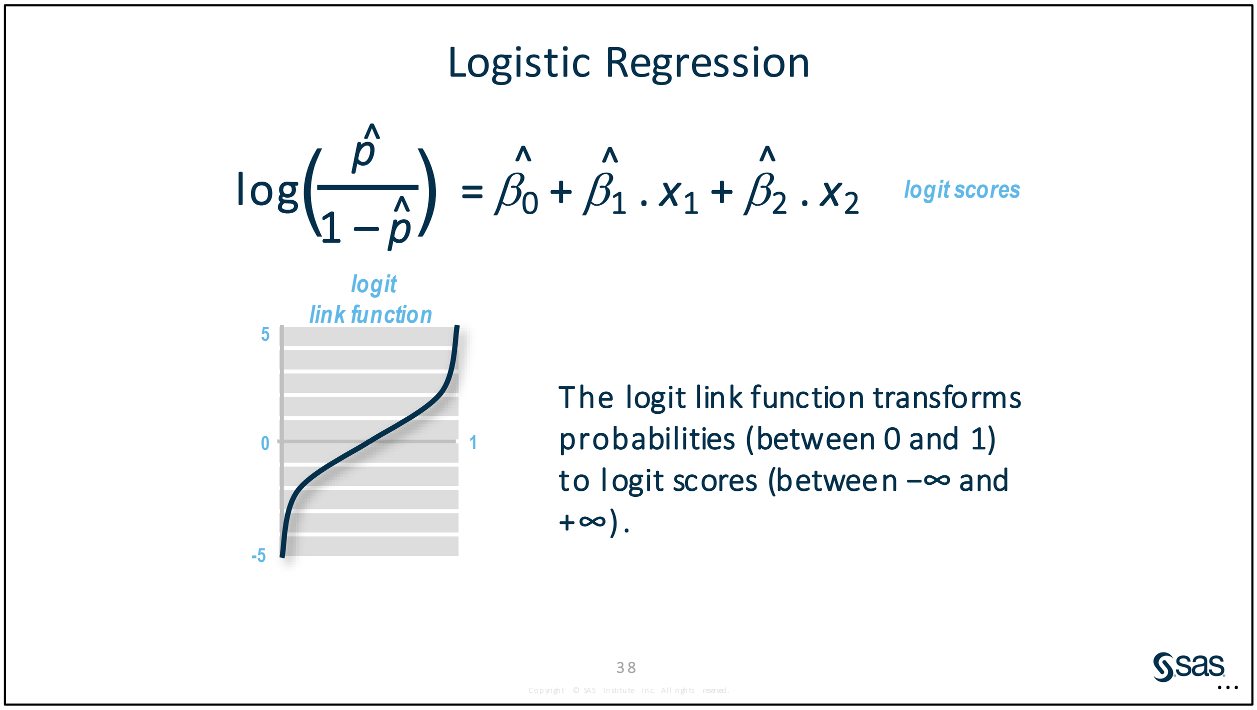
There are three levels of templates available, both for a class target as well as for an interval target. Intermediate template for class target was shown on the previous slide. You can create a new template from an existing pipeline, create a new template in the toolbox, and modify an existing template.

The advanced templates are also available with *autotuning* functionality. A large portion of the model-building process is taken up by experiments to identify the optimal set of parameters for the model algorithm. As algorithms get more complex (neural networks to deep neural networks, decision trees to forests and gradient boosting), the amount of time required to identify these parameters grows. There are several ways to support you in this cumbersome work of tuning machine learning model parameters. These approaches are called *hyperparameter optimization* and are discussed later in the course.

The following Pipeline templates are included with Model Studio:

Pipeline Template Name	Pipeline Template Description
Blank template	A data mining pipeline that contains only a Data node.
Basic template for class target	A simple linear flow: Data, Imputation, Logistic Regression, Model Comparison.
Basic template for interval target	A simple linear flow: Data, Imputation, Linear Regression, Model Comparison.
Intermediate template for class target	Extends the basic template with a stepwise logistic regression model and a decision tree.

Pipeline Template Name	Pipeline Template Description
Intermediate template for interval target	Extends the basic template with a stepwise linear regression model and a decision tree.
Advanced template for class target	Extends the intermediate template for class target with neural network, forest, and gradient boosting models, as well as an ensemble.
Advanced template for class target with autotuning	Advanced template for class target with autotuned tree, forest, neural network, and gradient boosting models.
Advanced template for interval target	Extends the intermediate template for interval target with neural network, forest, and gradient boosting models, as well as an ensemble.
Advanced template for interval target with autotuning	Advanced template for interval target with autotuned tree, forest, neural network, and gradient boosting models.
Automated feature engineering template	Template to perform automated feature engineering.



Logistic Regression

$$\log\left(\frac{\hat{p}}{1-\hat{p}}\right) = \hat{\beta}_0 + \hat{\beta}_1 \,.\, x_1 + \hat{\beta}_2 \,.\, x_2$$

logit scores

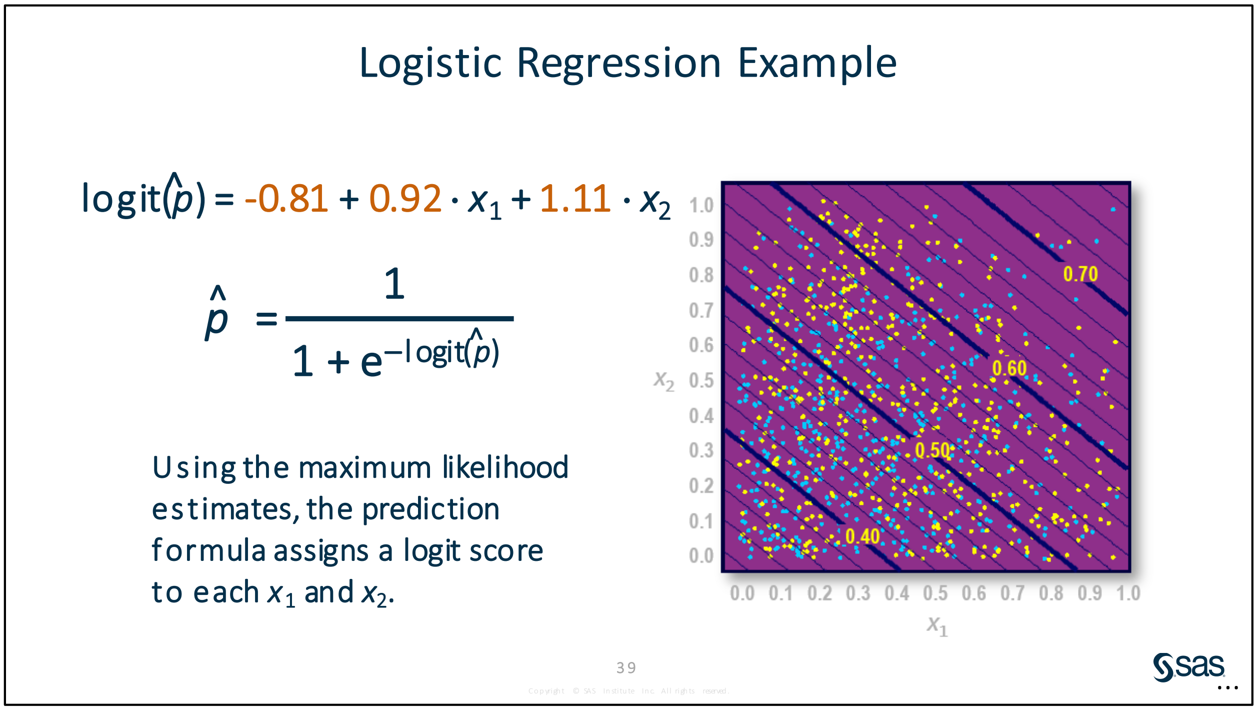
logit
link function
5
0
-5
1

The logit link function transforms probabilities (between 0 and 1) to logit scores (between −∞ and +∞).

38

In logistic regression, the expected value of the target is transformed by a link function to restrict its value to the unit interval. In this way, model predictions can be viewed as primary outcome probabilities. A linear combination of the inputs generates a *logit score*, the log of the odds of primary outcome, in contrast to the linear regression’s direct prediction of the target.

For binary prediction, any monotonic function that maps the unit interval to the real number line can be considered as a link. The logit link function is one of the most common. Its popularity is due, in part, to the interpretability of the model.



Logistic Regression Example

$$\text{logit}(\hat{p}) = -0.81 + 0.92 \cdot x_1 + 1.11 \cdot x_2$$

$$\hat{p} = \frac{1}{1 + e^{-\text{logit}(\hat{p})}}$$

Using the maximum likelihood estimates, the prediction formula assigns a logit score to each x_1 and x_2.

39

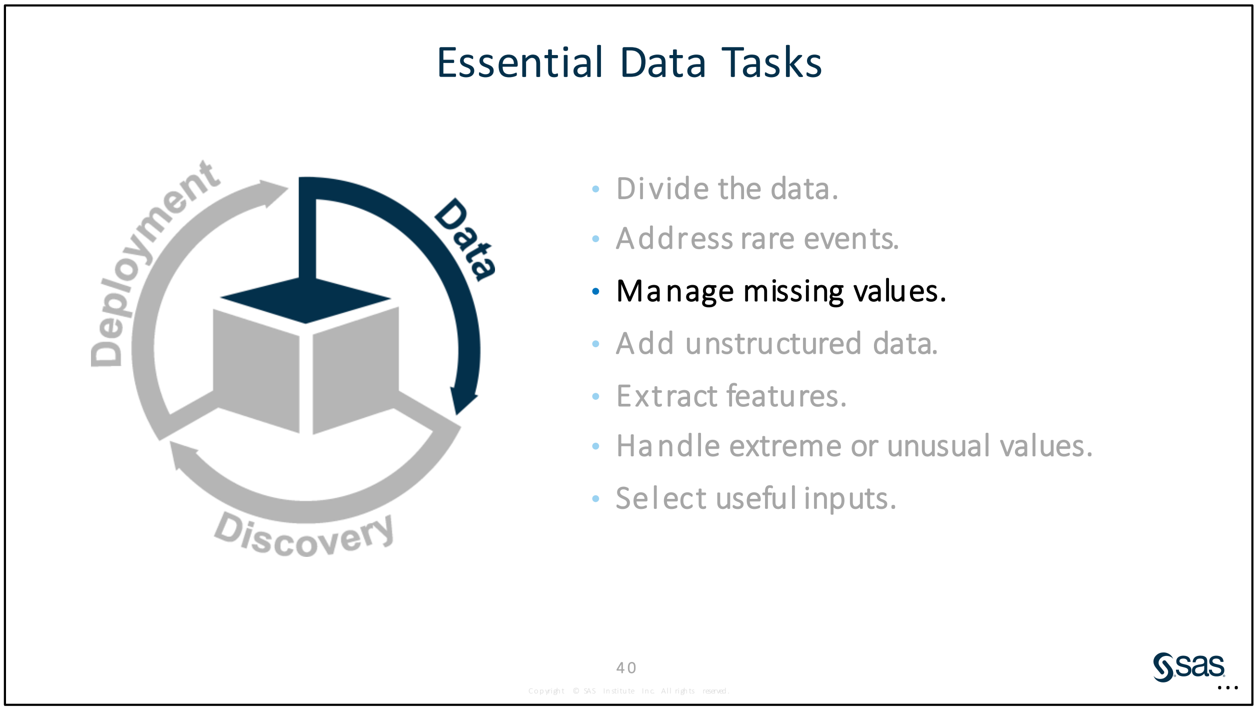
The presence of the logit link function complicates parameter estimation. Parameter estimates are obtained by maximum likelihood estimation. The likelihood function is the joint probability density of the data treated as a function of the parameters.

$$\sum log(\hat{p}_i) + \sum log(1 - \hat{p}_i)$$

The former quantity represents the primary outcome training cases and the later one represents the secondary outcome training cases, in the above expression.

The maximum likelihood estimates are the values of the parameters that maximize the probability of obtaining the training sample. These estimates can be used in the logit and logistic equations to obtain predictions. The plot on the right shows the prediction estimates from the logistic equation. One of the attractions of a standard logistic regression model is the simplicity of its predictions. The contours are simple straight lines commonly known as the *isoprobability lines*. (In higher dimensions, they would be hyperplanes.)

The predictions can be decisions, rankings, or estimates. The logit equation produces a ranking or logit score. To get a decision, you need a threshold. The easiest way to get a meaningful threshold is to convert the prediction ranking to a prediction estimate. You can obtain a prediction estimate using a straightforward transformation of the logit score, the logistic function. The *logistic function* is simply the inverse of the logit function. You can obtain the logistic function by solving the logit equation for *p*.



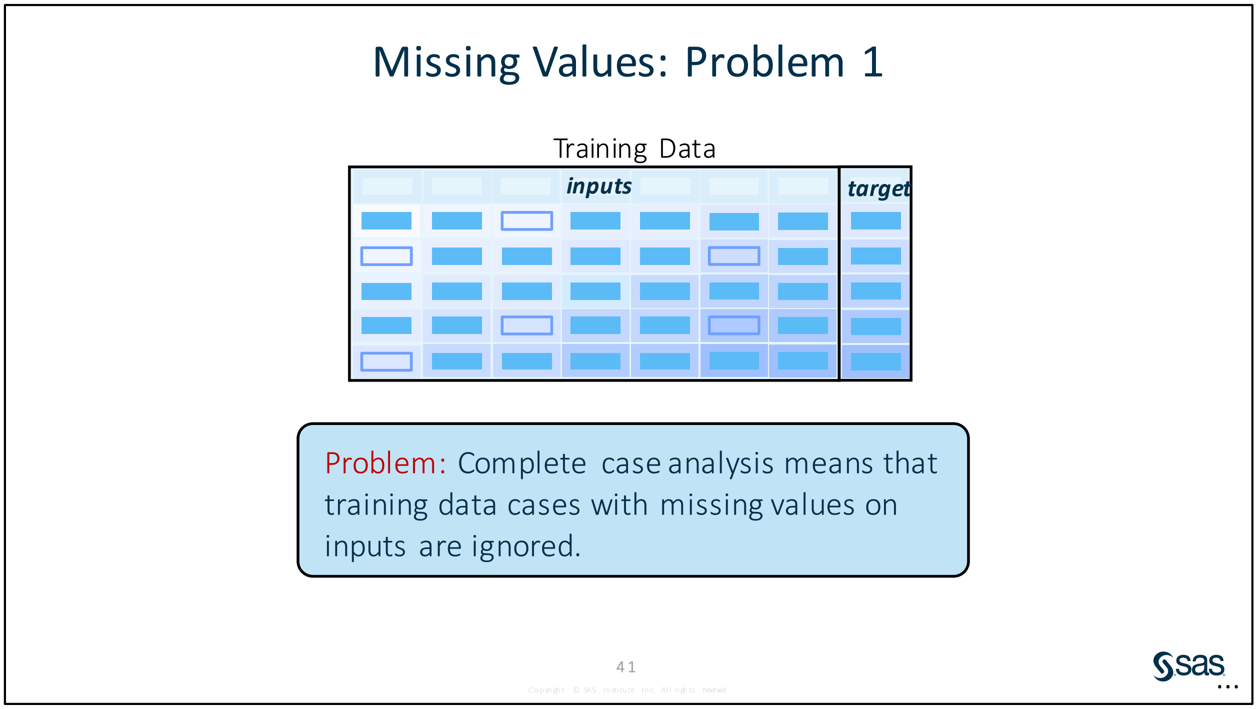
Essential Data Tasks

Deployment
Data
Discovery

- Divide the data.
- Address rare events.
- Manage missing values.
- Add unstructured data.
- Extract features.
- Handle extreme or unusual values.
- Select useful inputs.

40

Missing values can be theoretically and practically problematic for many machine learning tasks, especially when missing values are present in the target variable. This section addresses only the more common scenario of missing values in input variables.



Missing Values: Problem 1

Training Data

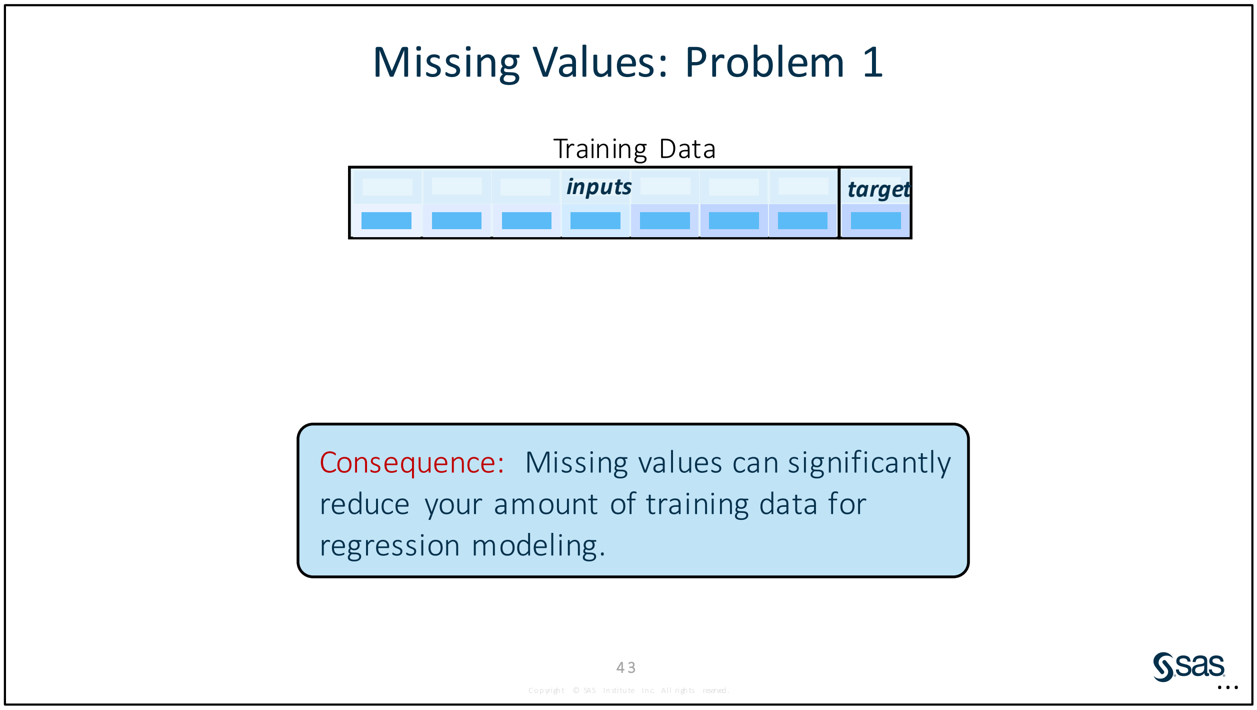
Problem: Complete case analysis means that training data cases with missing values on inputs are ignored.

41

The issue of missing values in data is (nearly) always present and always a concern. When faced with missing values in input variables, you must consider whether missing values are distributed randomly or whether missingness is somehow predictive of the target. If missing values appear at random in the input data, the input rows that contain missing values can be dropped from the analysis without introducing bias into the model. However, such a *complete case analysis* can remove a tremendous amount of information from the training data and reduce the predictive accuracy of the model. Many modeling algorithms in SAS Visual Data Mining and Machine Learning operate under complete case analysis (for example, linear and logistic regression, neural networks and support vector machines).

Complete case analysis assumes that data are missing completely at random and so does the Mean imputation. Imputation can be a more complicated issue, when missingness is nonrandom, dependent on inputs, or canonical. In this course, we use a simple approach that is often useful, but you should be aware that it is not always the best thing to do.

Missing values affect both model construction and model deployment.



Missing Values: Problem 1

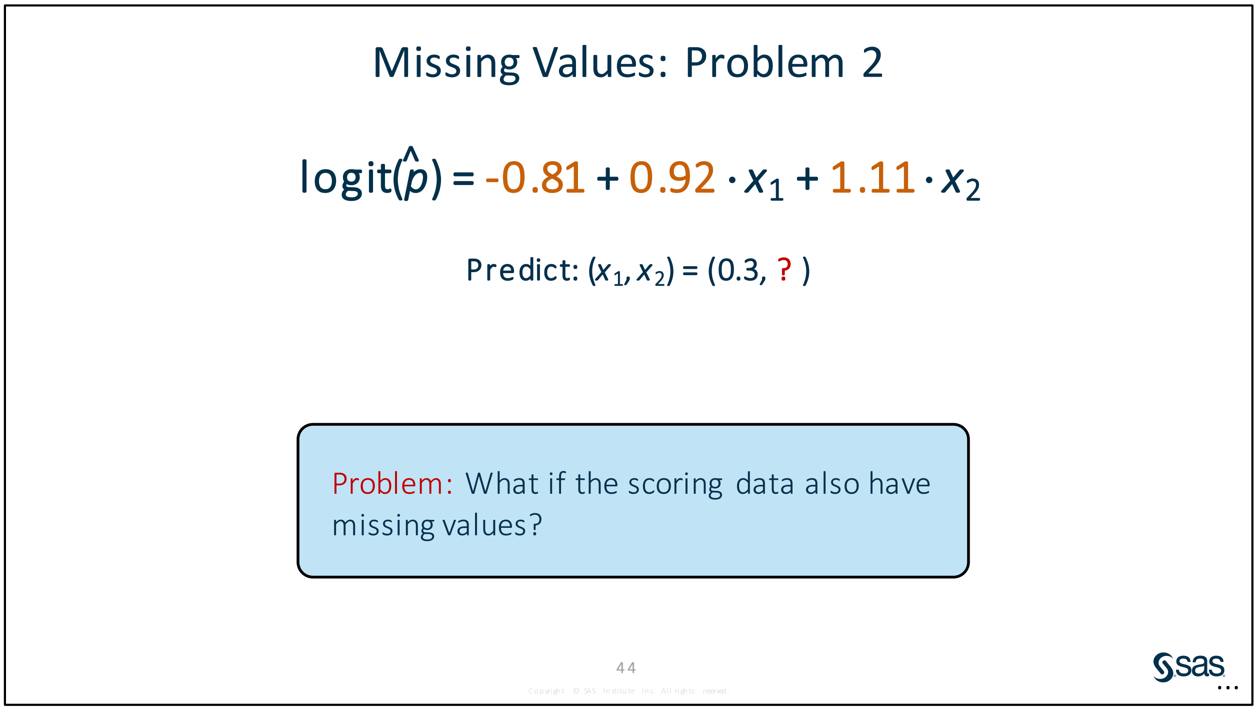
Consequence: Missing values can significantly reduce your amount of training data for regression modeling.

43

Even a smattering of missing values can cause an enormous loss of data in high dimensions. For example, suppose that each of the *k* input variables is missing at random with probability α. In this situation, the expected proportion of complete cases is as follows:

$$(1-\alpha)^k$$

Therefore, a 1% probability of missing (α=.01) for 100 inputs retains only 37% of the data for analysis, 200 keeps13%, and 400 preserves2%. If the “missingness” were increased to 5% (α=.05), then less than 1% of the data would be available with 100 inputs.



Missing Values: Problem 2

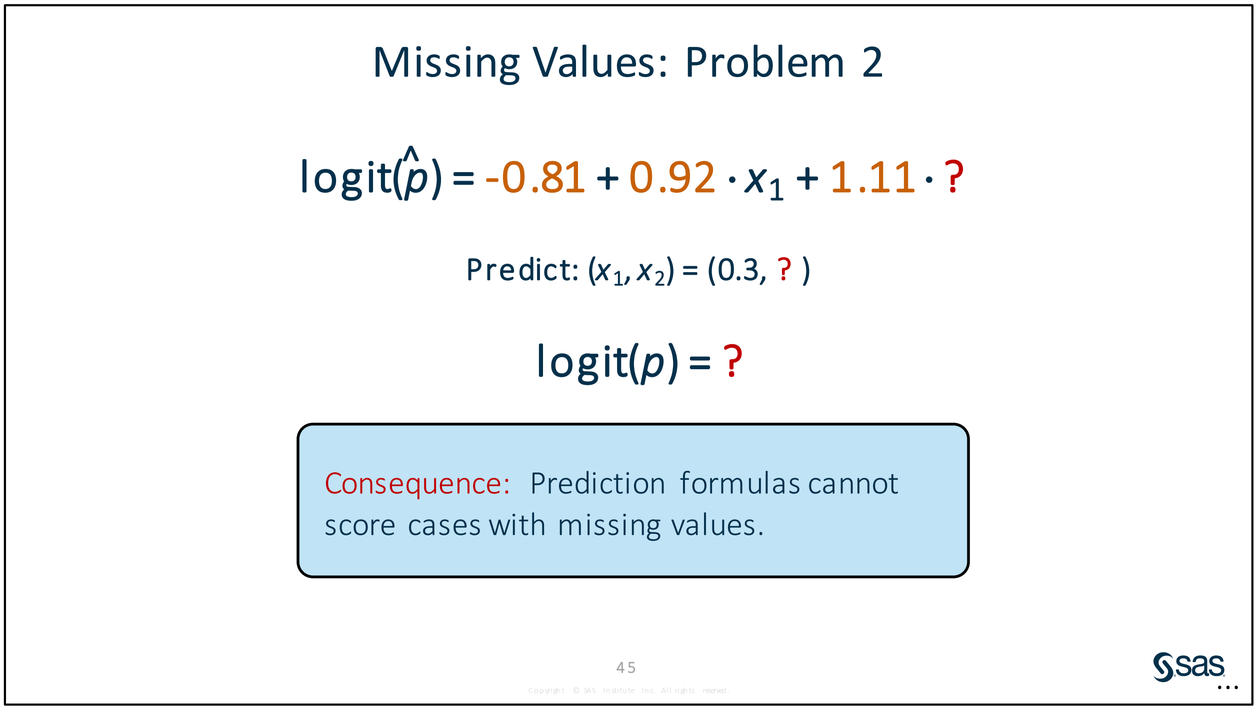
$$\mathrm{logit}(\hat{p}) = -0.81 + 0.92 \cdot x_1 + 1.11 \cdot x_2$$

Predict: $(x_1, x_2) = (0.3, ?)$

Problem: What if the scoring data also have missing values?

44

The second missing value problem relates to model deployment or using the prediction formula. How would a model built on the complete cases score a new case if it had a missing value? If there is missingness in your training data, it is very likely that your scoring data or the new data would also have, ideally, a similar type of missingness, but in limited amount.



Missing Values: Problem 2

$$\mathrm{logit}(\hat{p}) = -0.81 + 0.92 \cdot x_1 + 1.11 \cdot ?$$

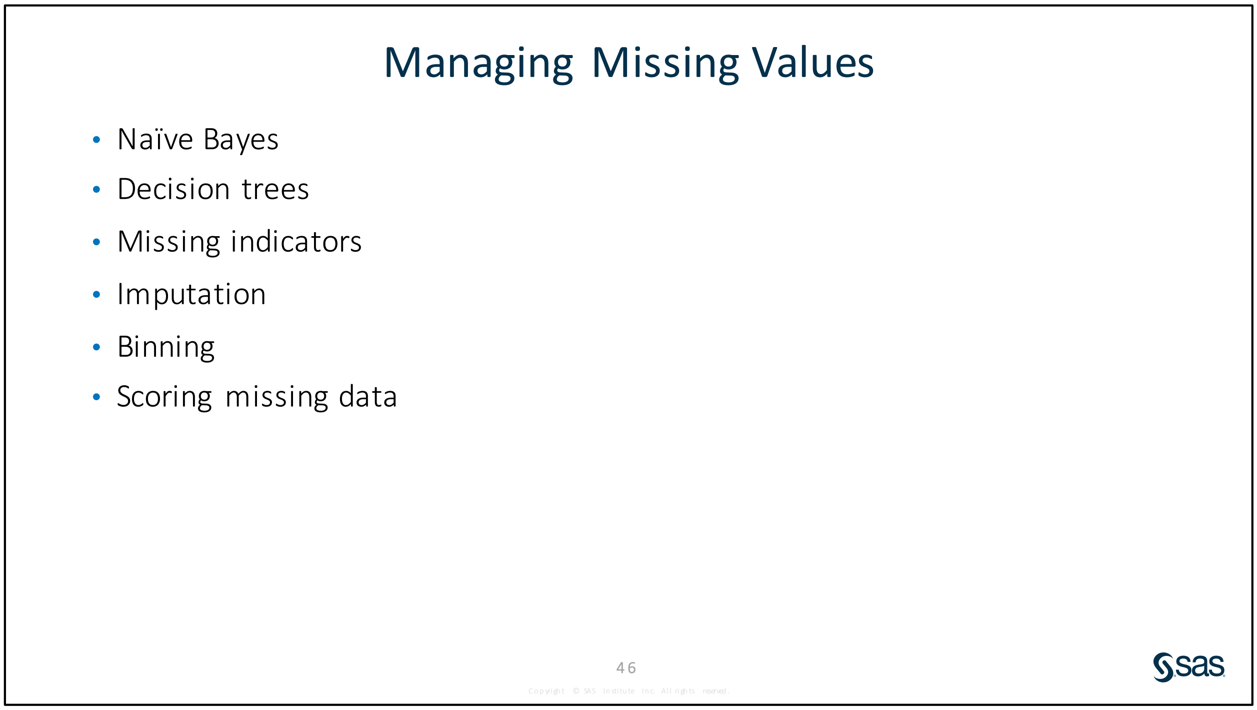
Predict: $(x_1, x_2) = (0.3, ?)$

$$\mathrm{logit}(p) = ?$$

Consequence: Prediction formulas cannot score cases with missing values.

45

A remedy is needed for the two problems of missing values. The appropriate remedy depends on the reason for the missing values.



Managing Missing Values

- Naïve Bayes
- Decision trees
- Missing indicators
- Imputation
- Binning
- Scoring missing data

46

Missingness can be predictive. Retaining information that is associated with missing values, including the missing values themselves, can increase the predictive accuracy of a model. The following list describes practices for accounting for missingness in training a machine learning model and describes how missing values must also be handled when scoring new data.

Naïve Bayes: Naïve Bayes models elegantly handle missing values for training and scoring by computing the likelihood based on the observed features. Because of conditional independence between the features, naïve Bayes ignores a feature only when its value is missing. Thus, you do not need to handle missing values before fitting a naïve Bayes model unless you believe that the missingness is not at random. For efficiency reasons, some implementations of naïve Bayes remove entire rows from the training process whenever a missing value is encountered. When missing is treated as a categorical level, infrequent missing values in new data can be problematic when they are not present in training data, because the missing level will have had no probability associated with it during training. You can solve this problem by ignoring the offending feature in the likelihood computation when scoring.

Decision trees: In general**,** imputation, missing markers, binning, and special scoring considerations are not required for missing values when you use a decision tree. Decision trees allow for the elegant and direct use of missing values in two common ways:

- When a splitting rule is determined, missing can be a valid input value, and missing values can either be placed on the side of the splitting rule that makes the best training prediction or be assigned to a separate branch in a split.
- Surrogate rules can be defined to allow the tree to split on a surrogate variable when a missing value is encountered. For example, a surrogate rule could be defined that allows a decision tree to split on the state variable when the ZIP code variable is missing.

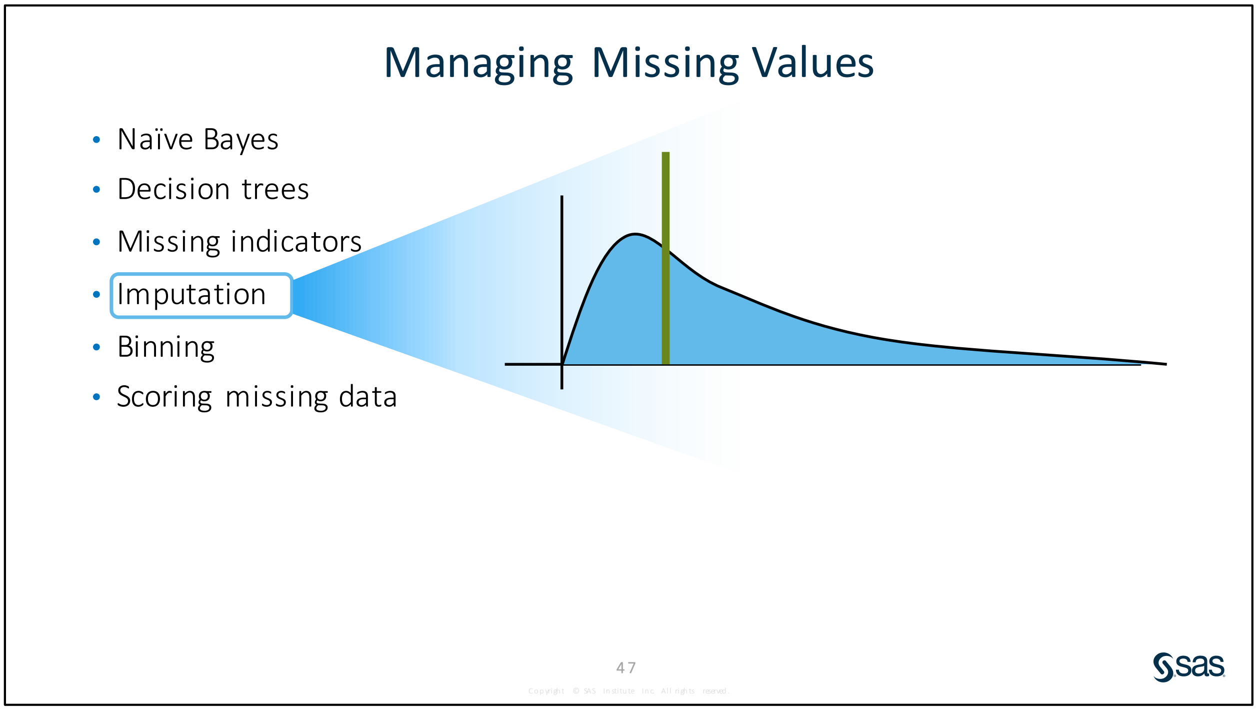
Missing markers: Missing markers are binary variables that record whether the value of another variable is missing. They are used to preserve information about missingness so that missingness can be modeled. Missing markers can be used in a model to replace the original corresponding variable with missing values, or they can be used in a model alongside an imputed version of the original variable.

Imputation: Imputation refers to replacing a missing value with information that is derived from nonmissing values in the training data. Simple imputation schemes include replacing a missing value in an input variable with the mean or mode of that variable's nonmissing values. For nonnormally distributed variables or variables that have a high proportion of missing values, simple mean or mode imputation can drastically alter a variable's distribution and negatively impact predictive accuracy. Even when variables are normally distributed and contain a low proportion of missing values, creating missing markers and using them in the model alongside the new, imputed variables is a suggested practice. Decision trees can also be used to derive imputed values. A decision tree can be trained using a variable that has missing values as its target and all the other variables in the data set as inputs. In this way, the decision tree can learn plausible replacement values for the missing values in the temporary target variable. This approach requires one decision tree for every input variable that has missing values, so it can become computationally expensive for large, dirty training sets. More sophisticated imputation approaches, including multiple imputation (MI), should be considered for small data sets (Rubin 1987).

Binning: Interval input variables that have missing values can be discretized into many bins according to their original numeric values to create new categorical, nominal variables. Missing values in the original variable can simply be added to an additional bin in the new variable. Categorical input variables that have missing values can be assigned to new categorical nominal variables that have the same categorical levels as the corresponding original variables plus one new level for missing values. Because binning introduces additional nonlinearity into a predictive model and can be less damaging to an input variable's original distribution than imputation, binning is generally considered acceptable, if not beneficial, until the binning process begins to contribute to overfitting. However, you might not want to use binning if the ordering of the values in an input variable is important, because the ordering information is changed or erased by introducing a missing bin into the otherwise ordered values.

Scoring missing data: If a decision tree or decision tree ensemble is used in training, missing values in new data will probably be scored automatically according to the splitting rules or the surrogate rules of the trained tree (or trees). If another type of algorithm was trained, then missing values in new data must be processed in the exact way that they were processed in the training data before the model was trained.

Source: *Best Practices for Machine Learning Applications* (Wujek, Hall, and Güneș 2016)
SAS Institute Inc.



Managing Missing Values

- Naïve Bayes
- Decision trees
- Missing indicators
- Imputation
- Binning
- Scoring missing data

47

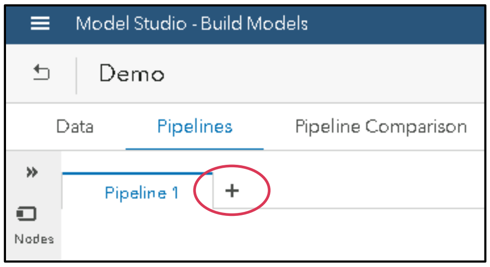
A remedy is needed for the two problems of missing values. The appropriate remedy depends on the reason for the missing values. In this course you will be focusing on imputing missing values.

In Model Studio, you can use a one-size-fits-all approach to handle missing values. In any case with a missing input measurement, the missing value is replaced with a fixed number. The net effect is to modify an input's distribution to include a point mass at the selected fixed number. The location of the point mass in synthetic distribution methods is not arbitrary. Ideally, it should be chosen to have minimal impact on the magnitude of an input's association with the target. With many modeling methods, this can be achieved by locating the point mass at the input's mean value.

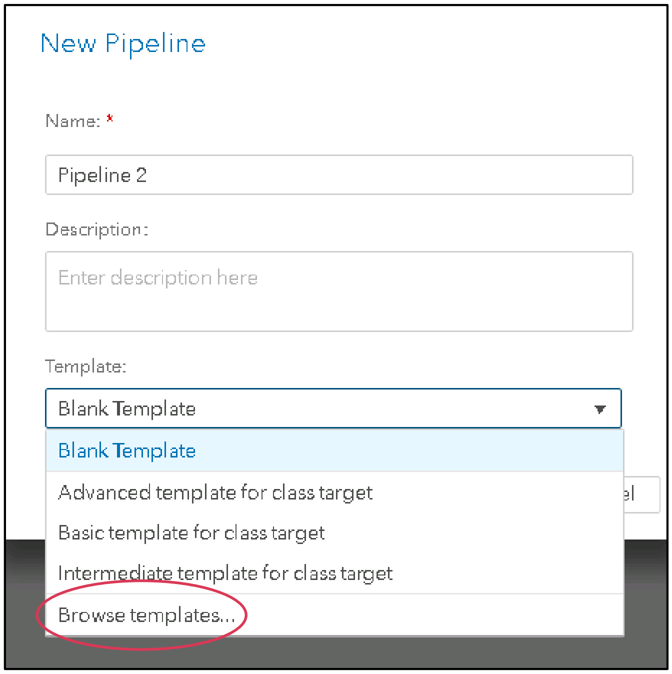
Building a Pipeline from a Basic Template

Although it is nice to be able to build up your own pipelines from scratch, it is often convenient to start from a template that represents best practices in building predictive models. The application comes with a nice set of templates available for creating new pipelines. In this demonstration, to start simple, you build a new pipeline from a basic template for class target.

1. Click + next to the current pipeline tab in the upper left corner of the canvas.



2. In the New Pipeline window, select **Browse templates** under Template.



New Pipeline

Name: * Pipeline 2

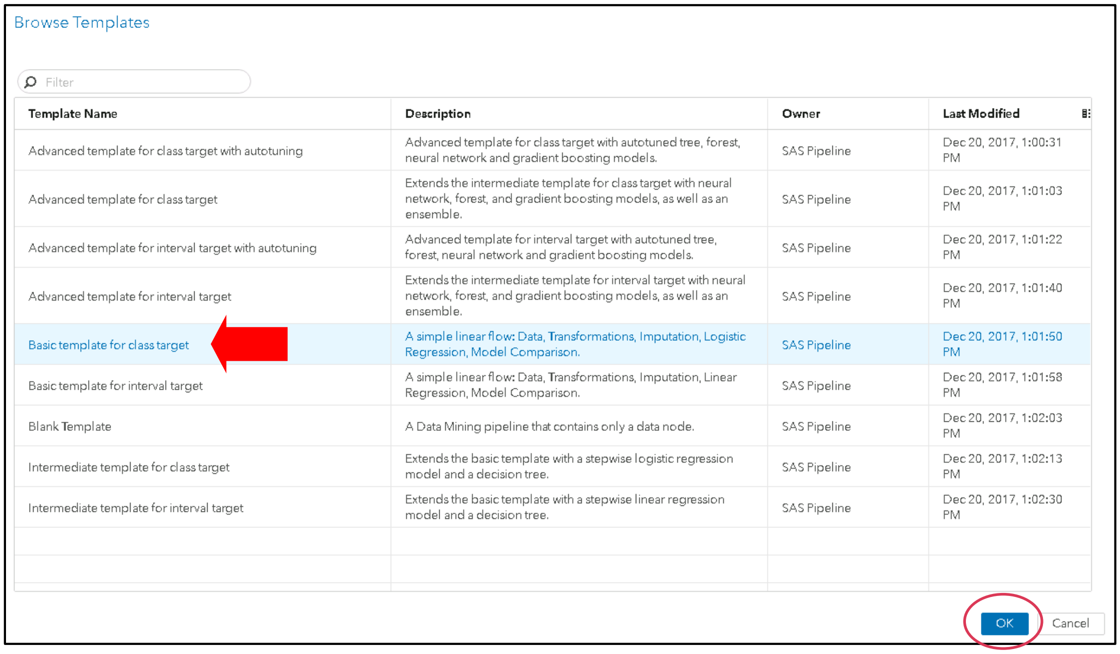
Description: Enter description here

Template: Blank Template

Blank Template
Advanced template for class target
Basic template for class target
Intermediate template for class target
Browse templates...

Note: Some of the options on the Templates menu might be different on your classroom computer from what is shown above.

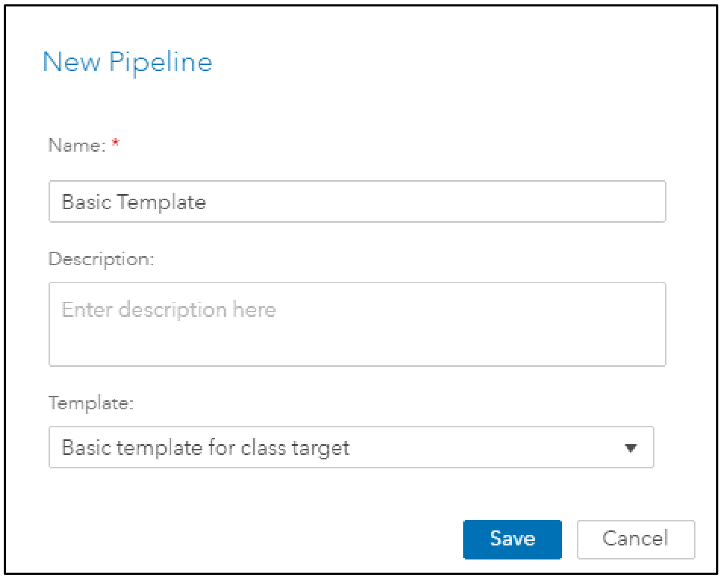
3. In the Browse Templates window, select **Basic template for class target**. Click **OK**.



Browse Templates

Template Name	Description	Owner	Last Modified
Advanced template for class target with autotuning	Advanced template for class target with autotuned tree, forest, neural network and gradient boosting models.	SAS Pipeline	Dec 20, 2017, 1:00:31 PM
Advanced template for class target	Extends the intermediate template for class target with neural network, forest, and gradient boosting models, as well as an ensemble.	SAS Pipeline	Dec 20, 2017, 1:01:03 PM
Advanced template for interval target with autotuning	Advanced template for interval target with autotuned tree, forest, neural network and gradient boosting models.	SAS Pipeline	Dec 20, 2017, 1:01:22 PM
Advanced template for interval target	Extends the intermediate template for interval target with neural network, forest, and gradient boosting models, as well as an ensemble.	SAS Pipeline	Dec 20, 2017, 1:01:40 PM
Basic template for class target	A simple linear flow: Data, Transformations, Imputation, Logistic Regression, Model Comparison.	SAS Pipeline	Dec 20, 2017, 1:01:50 PM
Basic template for interval target	A simple linear flow: Data, Transformations, Imputation, Linear Regression, Model Comparison.	SAS Pipeline	Dec 20, 2017, 1:01:58 PM
Blank Template	A Data Mining pipeline that contains only a data node.	SAS Pipeline	Dec 20, 2017, 1:02:03 PM
Intermediate template for class target	Extends the basic template with a stepwise logistic regression model and a decision tree.	SAS Pipeline	Dec 20, 2017, 1:02:13 PM
Intermediate template for interval target	Extends the basic template with a stepwise linear regression model and a decision tree.	SAS Pipeline	Dec 20, 2017, 1:02:30 PM

4. In the New Pipeline window, name the pipeline **Basic Template**.



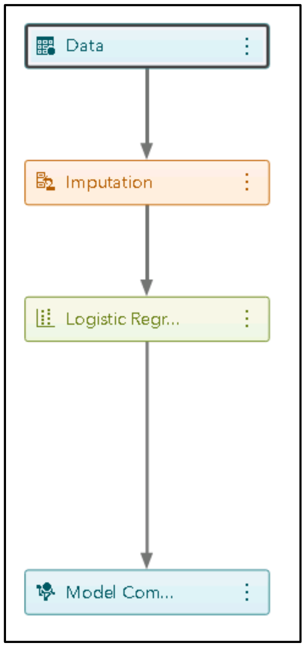
New Pipeline

Name: * Basic Template

Description: Enter description here

Template: Basic template for class target

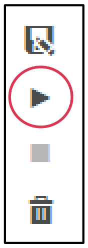
5. Click **Save**.



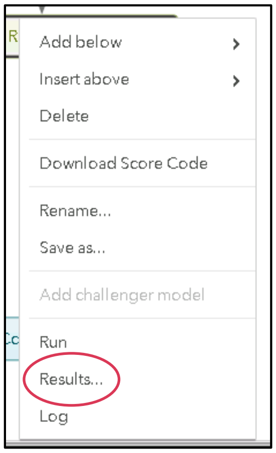
The *basic template for class target* is a simple linear flow and includes the following nodes: Data, Imputation, Logistic Regression, and Model Comparison. You can add additional nodes by right-clicking the existing nodes (or dragging and dropping from the Nodes pane.)

Note: Because a predicted response might be different for cases with a missing input value, a binary imputation indicator variable is often added to the training data. Adding this variable enables a model to adjust its predictions in the situation where "missingness" itself is correlated with the target.

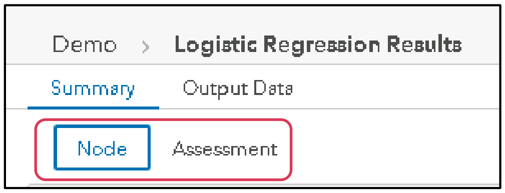
6. Click the **Run pipeline** icon.



7. After the pipeline has successfully run, right-click the **Logistic Regression** node and select **Results**.



The Results window contains two important tabs at the top: one for Node results and one for Assessment results.

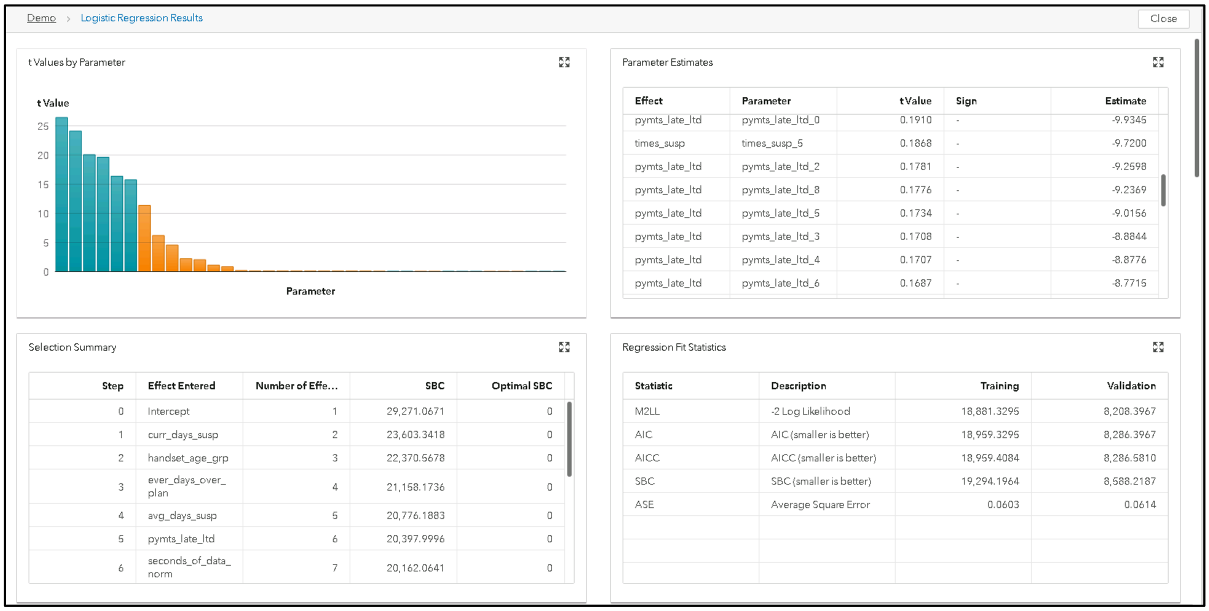


Here are some of the windows included under the Nodes tab in the results from the Logistic Regression node:

- t-values by Parameter table
- Parameter Estimates table
- Selection Summary table

Here are some of the windows included under the Assessment tab in the results from the Logistic Regression node:

- Lift Reports plots
- Fit Statistics table
- Output



Parameter Estimates

Effect	Parameter	t Value	Sign	Estimate
pymts_late_ltd	pymts_late_ltd_0	0.1910	-	-9.9345
times_susp	times_susp_5	0.1868	-	-9.7200
pymts_late_ltd	pymts_late_ltd_2	0.1781	-	-9.2598
pymts_late_ltd	pymts_late_ltd_8	0.1776	-	-9.2369
pymts_late_ltd	pymts_late_ltd_5	0.1734	-	-9.0156
pymts_late_ltd	pymts_late_ltd_3	0.1708	-	-8.8844
pymts_late_ltd	pymts_late_ltd_4	0.1707	-	-8.8776
pymts_late_ltd	pymts_late_ltd_6	0.1687	-	-8.7715

Selection Summary

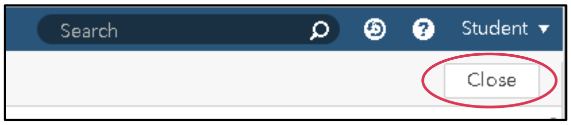
Step	Effect Entered	Number of Effe...	SBC	Optimal SBC
0	Intercept	1	29,271.0671	0
1	curr_days_susp	2	23,603.3418	0
2	handset_age_grp	3	22,370.5678	0
3	ever_days_over_plan	4	21,158.1736	0
4	avg_days_susp	5	20,776.1883	0
5	pymts_late_ltd	6	20,397.9996	0
6	seconds_of_data_norm	7	20,162.0641	0

Regression Fit Statistics

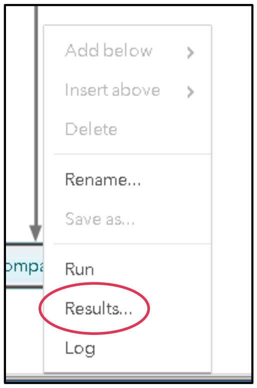
Statistic	Description	Training	Validation
M2LL	-2 Log Likelihood	18,881.3295	8,208.3967
AIC	AIC (smaller is better)	18,959.3295	8,286.3967
AICC	AICC (smaller is better)	18,959.4084	8,286.5810
SBC	SBC (smaller is better)	19,294.1964	8,588.2187
ASE	Average Square Error	0.0603	0.0614

Explore the results as you see fit.

8. Close the Results window by clicking **Close** in the upper right corner of the window.

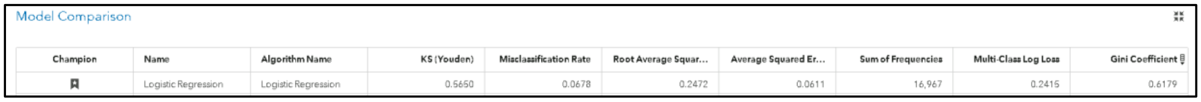


9. Right-click the **Model Comparison** node and select **Results**.



10. Click ⛶ to expand the Model Comparison table. Unless specified, the default fit statistic (KS) is used for selecting a champion model with a class target.

Note: To change the default fit statistic for just this comparison, change the class selection statistic of the Model Comparison properties. To change the default fit statistic for all projects, change the class selection statistic on the Project Settings menu. The default is the Kolmogorov-Smirnov statistic (KS).



Model Comparison

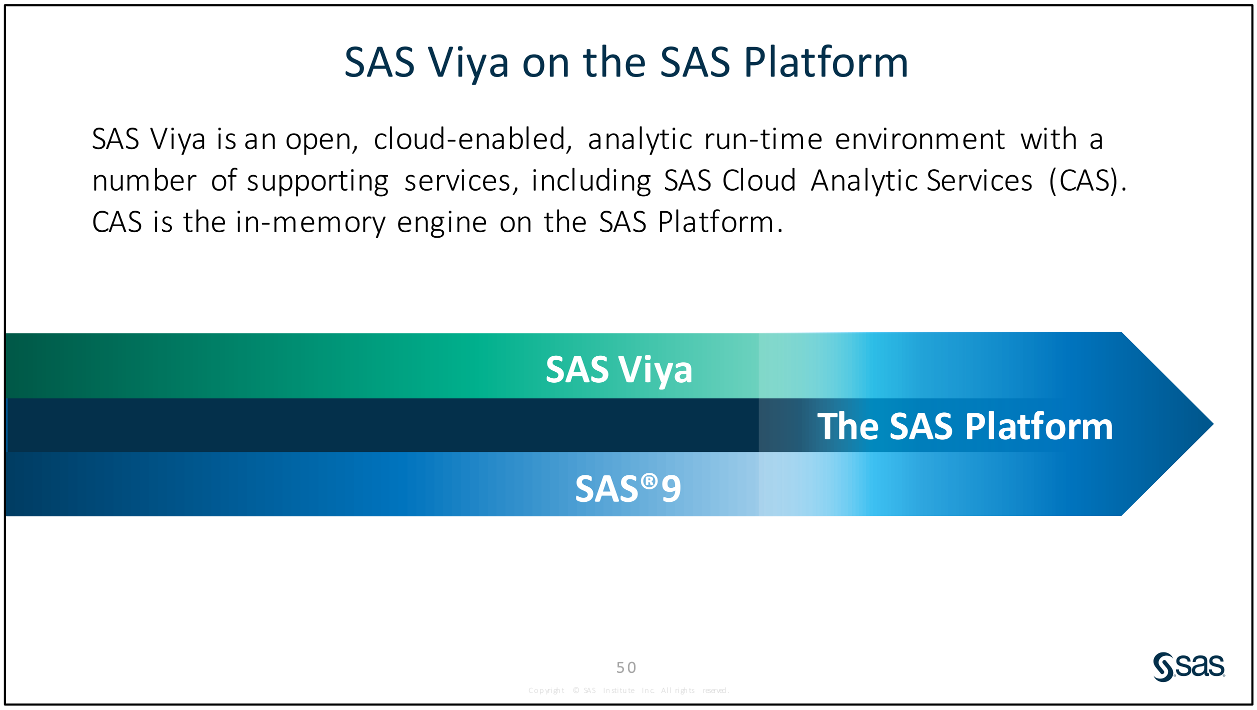
Champion	Name	Algorithm Name	KS (Youden)	Misclassification Rate	Root Average Squar...	Average Squared Er...	Sum of Frequencies	Multi-Class Log Loss	Gini Coefficient
🔖	Logistic Regression	Logistic Regression	0.5650	0.0678	0.2472	0.0611	16,967	0.2415	0.6179

Note: The Model Comparison node is always added by default when any model is contained in the pipeline. If the pipeline contains only a single model, the Model Comparison node summarizes performance of this one model.

11. Click to exit the maximized view.
12. Click **Close** to close the Model Comparison Results window.

End of Demonstration

1.3 Introduction to SAS Viya



SAS Viya on the SAS Platform

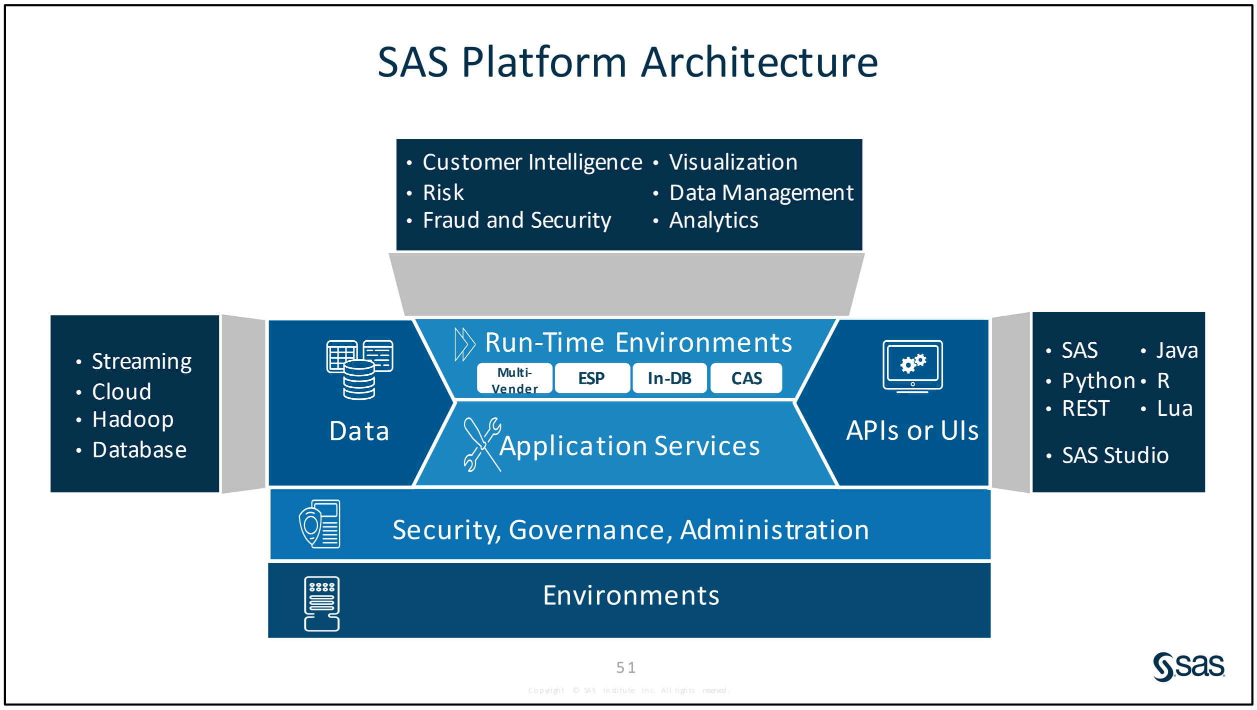
SAS Viya is an open, cloud-enabled, analytic run-time environment with a number of supporting services, including SAS Cloud Analytic Services (CAS). CAS is the in-memory engine on the SAS Platform.

SAS Viya

The SAS Platform

SAS®9

50



SAS Platform Architecture

- Customer Intelligence
- Risk
- Fraud and Security
- Visualization
- Data Management
- Analytics

- Streaming
- Cloud
- Hadoop
- Database

Data

Run-Time Environments

Multi-Vender | ESP | In-DB | CAS

Application Services

APIs or UIs

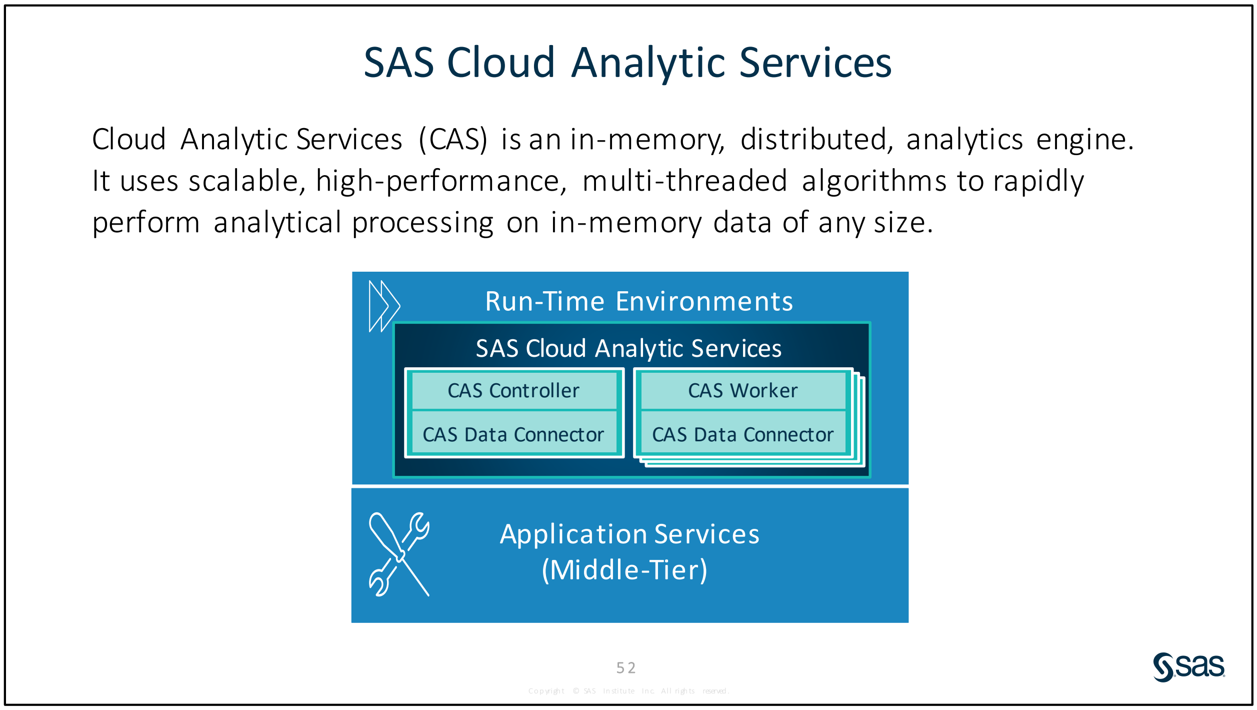
- SAS
- Python
- REST
- Java
- R
- Lua
- SAS Studio

Security, Governance, Administration

Environments

51

The SAS Analytics Platform is a software foundation that is engineered to generate insights from your data in any computing environment. Built on a strategy of using analytical insights to drive business actions, this platform supports every phase of the analytics life cycle from data, to discovery, to deployment.



SAS Cloud Analytic Services

Cloud Analytic Services (CAS) is an in-memory, distributed, analytics engine. It uses scalable, high-performance, multi-threaded algorithms to rapidly perform analytical processing on in-memory data of any size.

Run-Time Environments

SAS Cloud Analytic Services

CAS Controller

CAS Data Connector

CAS Worker

CAS Data Connector

Application Services (Middle-Tier)

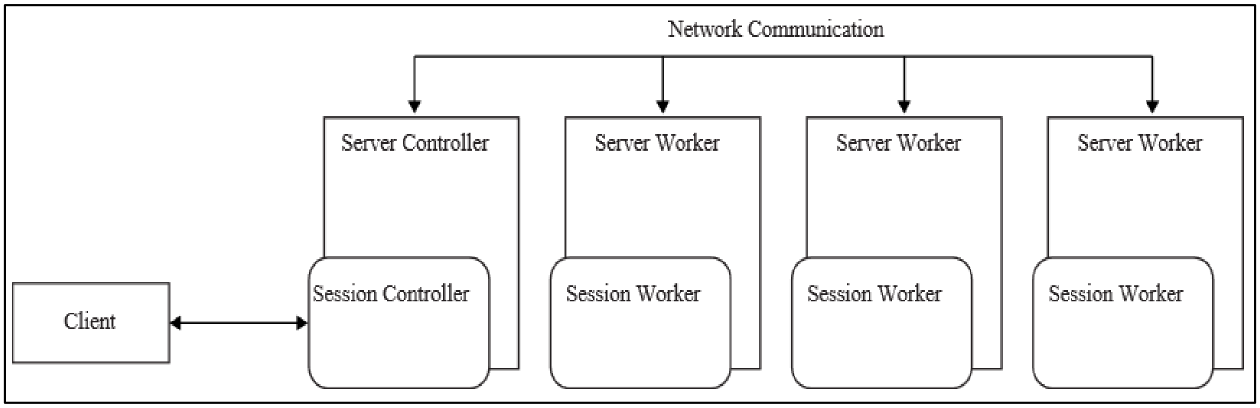
52

Run-time environment refers to the combination of hardware and software in which data management and analytics occur.

CAS is designed to run in a single-machine symmetric multiprocessing (SMP) or multi-machine massively parallel processing (MPP) configuration. CAS supports multiple platform and infrastructure configurations.

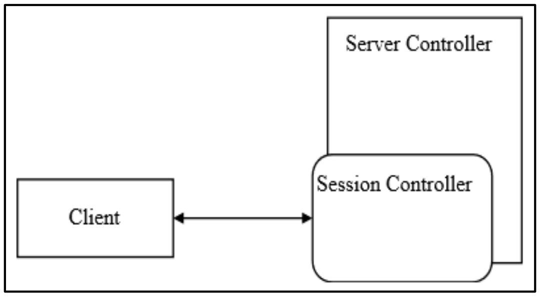
CAS also has a communications layer that supports fault tolerance. When CAS is running in an MPP configuration, it can continue processing requests even if it loses connectivity to some nodes. This communication layer also enables you to remove or add nodes while the server is running.

Distributed Server: Massively Parallel Processing (MPP)



A distributed server uses multiple machines to perform massively parallel processing. The figure in the slide above depicts the server topology for a distributed server. Of the multiple machines used, one machine acts as the controller and other machines act as workers to process data. Client applications communicate with the controller, and the controller coordinates the processing that is performed by the worker nodes. One or more machines are designated as worker nodes. Each worker node performs data analysis on the rows of data that are in-memory on the node. The server scales horizontally. If processing times are unacceptably long due to large data volumes, more machines can be added as workers to distribute the workload. Distributed servers are fault tolerant. If communication with a worker node is lost, a surviving worker node uses a redundant copy of the data to complete the data analysis. Whenever possible, distributed servers load data into memory in parallel. This provides the fastest load times.

Single-Machine Server: Symmetric Multiprocessing (SMP)

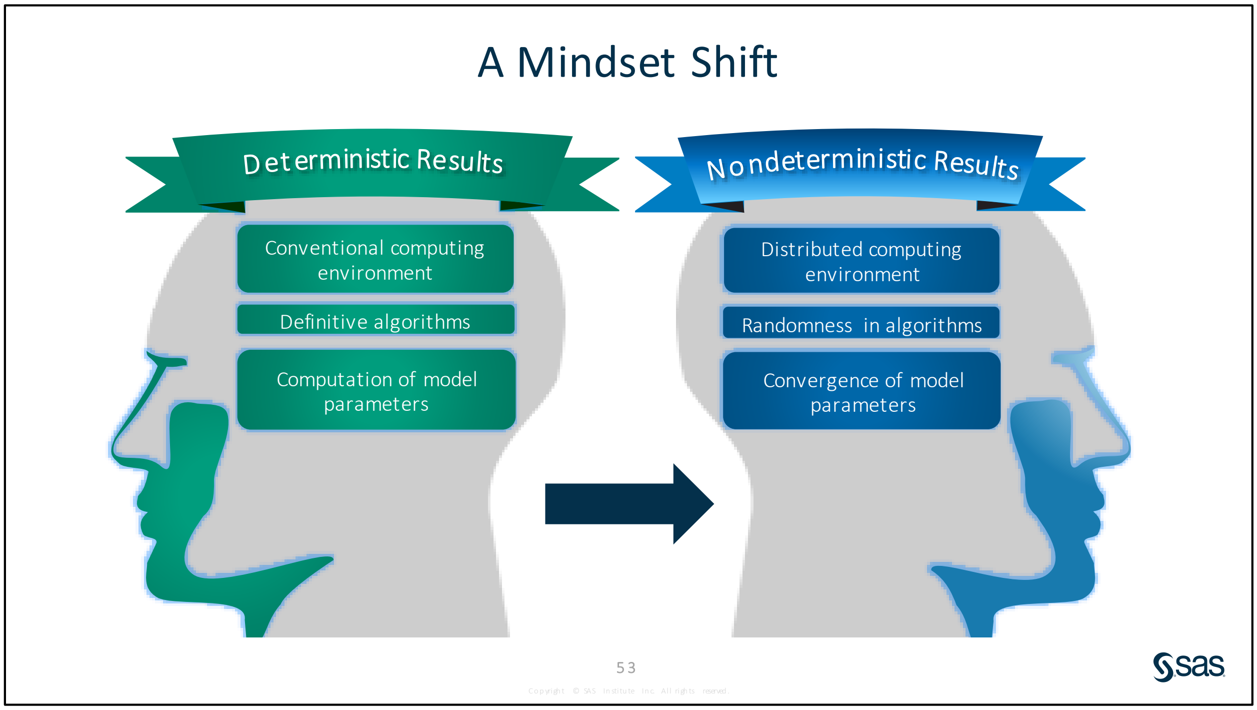


The figure above depicts the server topology for a single-machine server. The single machine is designated as the controller. Because there are no worker nodes, the controller node performs data analysis on the rows of data that are in-memory. The single machine uses multiple CPUs and threads to speed up data analysis. This architecture is often referred to as symmetric multi-processing, or SMP. All the in-memory analytic features of a distributed server are available to the single-machine server. Single-machine servers cannot load data into memory in parallel from any data source.

Leveraging the CAS server that is part of the SAS Viya release includes a whole host of tangible benefits. The main reason is represented by a simple three-word phrase: tremendous performance gains. Because processes run so much faster, you can complete your work faster. This means that you can complete more work, and even entire projects, in a significantly reduced time frame.

Processing Type	Multi-threaded, Single Machine (SAS Viya SMP)	Multi-threaded, Multiple Machines (SAS Viya MPP)
Distributed, parallel processing?	Yes	Yes
In-memory data persistence?	Yes	Yes
Common performance speed-up	10x – 20x	Up to 100x*

* Increase depends on many factors including hardware allocation; performance could be higher.



A Mindset Shift

Deterministic Results

- Conventional computing environment
- Definitive algorithms
- Computation of model parameters

Nondeterministic Results

- Distributed computing environment
- Randomness in algorithms
- Convergence of model parameters

53

In SAS Viya, you might have nondeterministic results or might not get the reproducible results, essentially because of two reasons:

1. distributed computing environment
2. nondeterministic algorithms

In distributed computing, cases are divided over compute nodes and there could be variation in the results. You might get slightly different results even in the same server when the controllers/workers are more manageable. In different servers, this is even more expectable. CAS server represents pooled memory and runs code multi-threaded. Multi-threading tends to distribute the same instructions to other available threads for execution creating many different queues on many different cores using separate allocations or subsets of data. Most of the time, multiple threads perform operations on isolated collections of data that are independent of one another, but part of a larger table. For that reason, it is possible to have a counter (for example, n+1;) operating on one thread to produce a result that might be different from a counter operating on another thread, because each thread is working on a different subset of the data. Therefore, results can be different from thread to thread unless and until the individual results from multiple threads are summed together. It is not as complicated as it might sound. That is because SAS Viya automatically takes care of most collation and reassembly of processing results, with a few minor exceptions where you must further specify how to combine results from multiple threads.

A nondeterministic algorithm is an algorithm that, even for the same input, can exhibit different behaviors on different runs, as opposed to a deterministic algorithm. There are several ways an algorithm might behave differently from run to run. A concurrent algorithm can perform differently on different runs due to a race condition. A probabilistic algorithm's behaviors depend on a random number generator. The nondeterministic algorithms are often used to find an approximation to a solution when the exact solution would be too costly to obtain using a deterministic one (Wikipedia). Some SAS Visual Data Mining and Machine Learning models are created with a nondeterministic process. This means that you might experience different displayed results when you run a model, save that model, close the model, and re-open the report or print the report later.

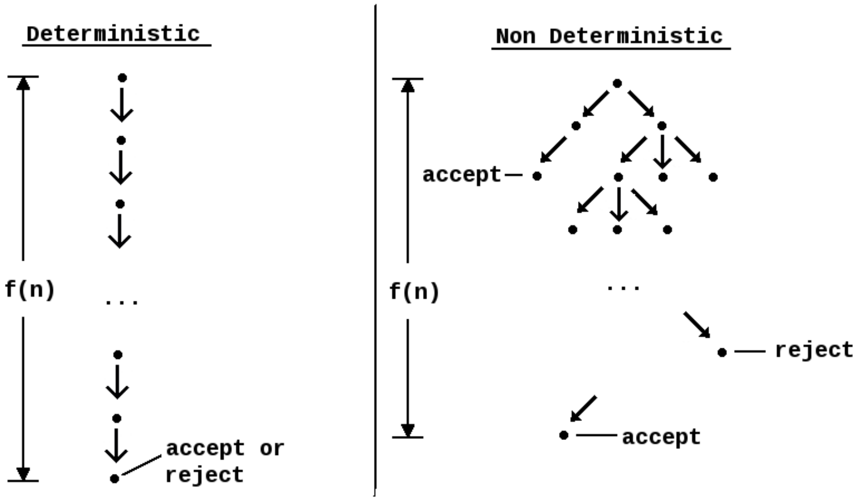


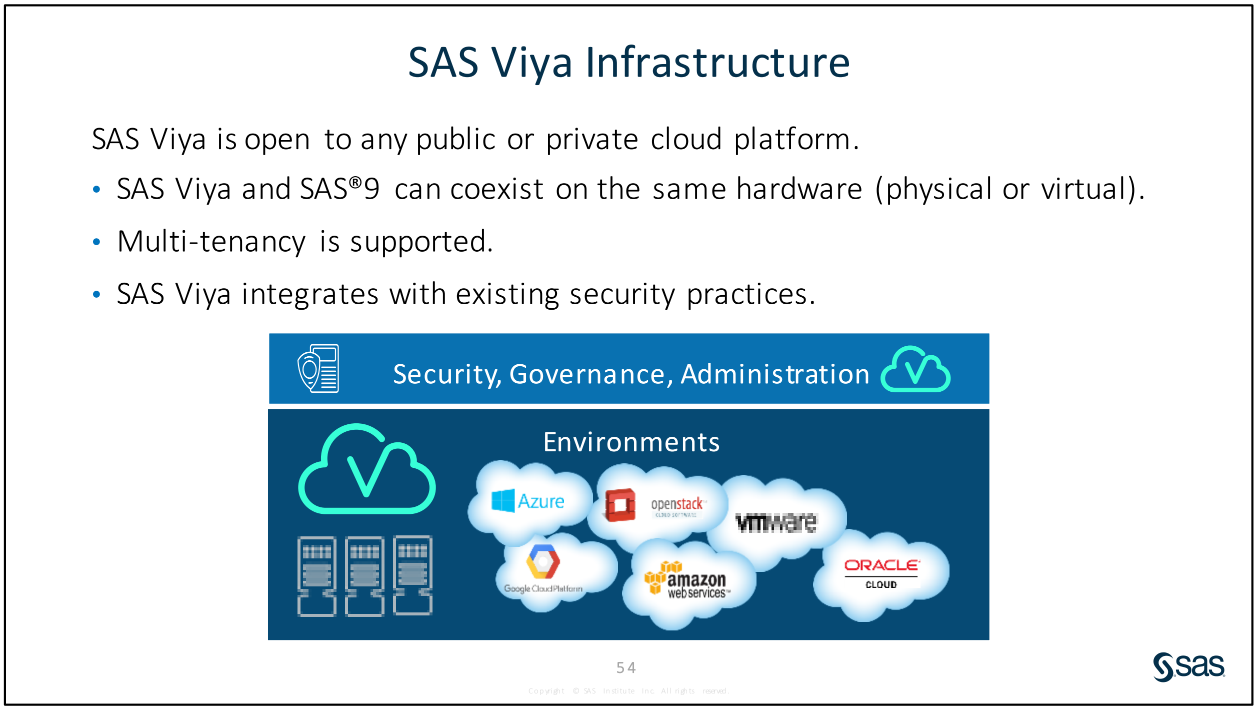
Image source: By Eleschinski2000 - With a paint program., CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=43528132

A deterministic algorithm that performs *f*(*n*) steps always finishes in *f*(*n*) steps and always returns the same result. A nondeterministic algorithm that has *f*(*n*) levels might not return the same result on different runs. A nondeterministic algorithm might never finish due to the potentially infinite size of the fixed height tree.

It is an altogether different mindset!

You are “converging” on a model or “estimating” a model, not exactly computing the parameters of the model. Bayesians understand this, when they look for convergence of parameters. They try to converge to a distribution, not a point. Maybe it would be interesting to try running the models 10 times across different samples and ensembling them to see the dominant signal. You cannot expect the results to be reproduced because some algorithms have randomness included in the process.

However, the results do converge. This is a distinguished computing environment designed for big data, and this non-reproducibility is the price we pay.



SAS Viya Infrastructure

SAS Viya is open to any public or private cloud platform.

- SAS Viya and SAS®9 can coexist on the same hardware (physical or virtual).
- Multi-tenancy is supported.
- SAS Viya integrates with existing security practices.

Security, Governance, Administration
Environments
Azure
openstack
vmware
Google Cloud Platform
amazon web services
ORACLE CLOUD

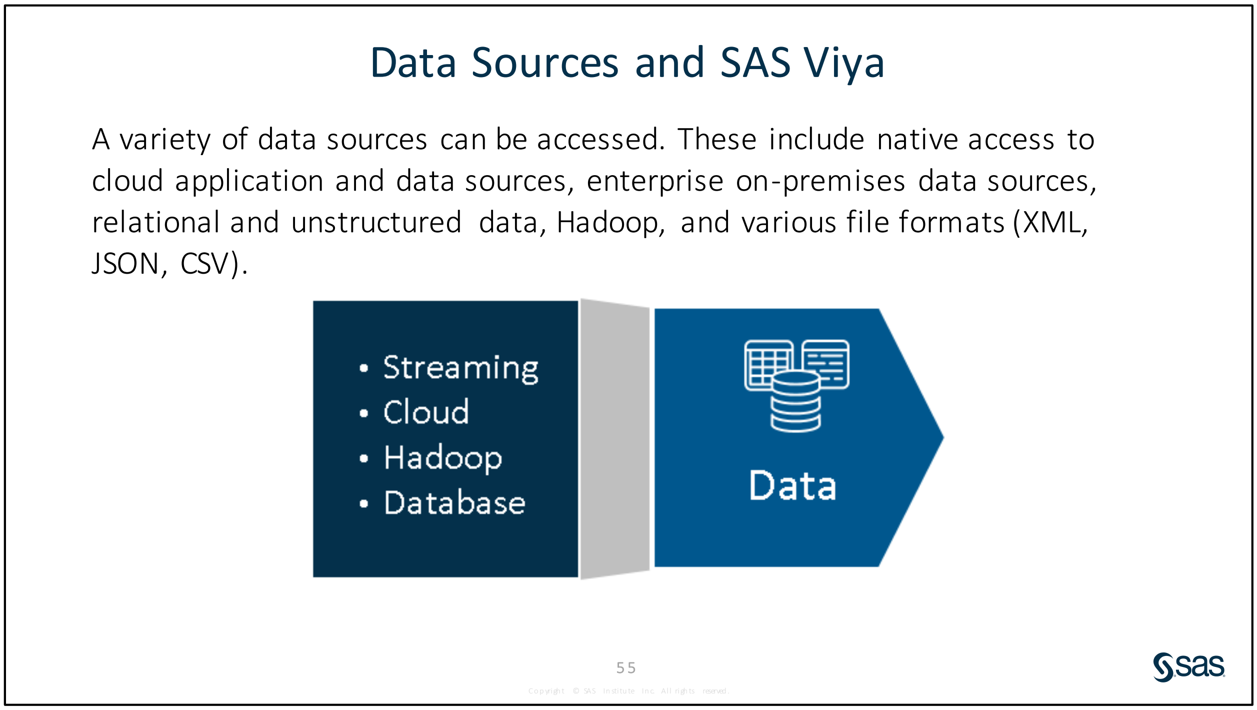
54

SAS moved from multi-vendor to multi-cloud support. The IT Department can add compute resources from whom they want and when they need them.

Each tenant is completely isolated from other tenants. For example, a use-case would be a tenant for your Human Resources Department. You must guarantee that access to sensitive data is restricted to appropriate personnel.

Activating multi-tenancy is a deployment-time decision. Ensure that you understand the implications of this choice before you deploy resources.

SAS Viya is deployed with Transport Layer Security (TLS) to secure network connections and is fully compliant with SAS security standards.



Data Sources and SAS Viya

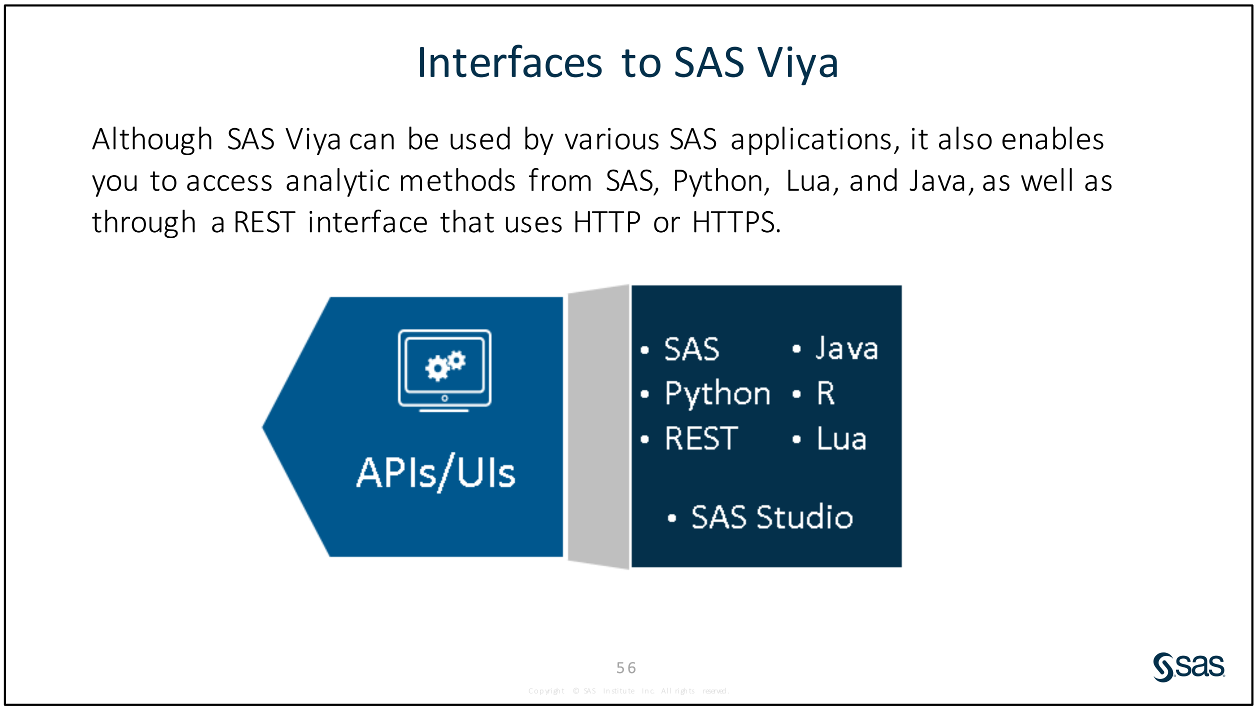
A variety of data sources can be accessed. These include native access to cloud application and data sources, enterprise on-premises data sources, relational and unstructured data, Hadoop, and various file formats (XML, JSON, CSV).

- Streaming
- Cloud
- Hadoop
- Database

Data

55

Persisted in-memory data is another unique SAS Viya feature. This is the shift away from using flat files and external data sets to a strategy of using persisted, pre-loaded data tables. In SAS Viya, all data typically go through an I/O conversion process only once and can be reused as many times as needed thereafter, without incurring the same expense of conversion into a binary, machine-level format. SAS Viya data is either stored within the RAM of a single machine (and runs in SMP mode) or within a shared pool of allocated memory created from several networked machines as part of a common memory grid (which enables Massively Parallel Processing, or MPP mode). That pooled memory array is an integral part of CAS. After the data is loaded into CAS, all processing instructions execute very quickly against the pre-converted, in-memory data.



Interfaces to SAS Viya

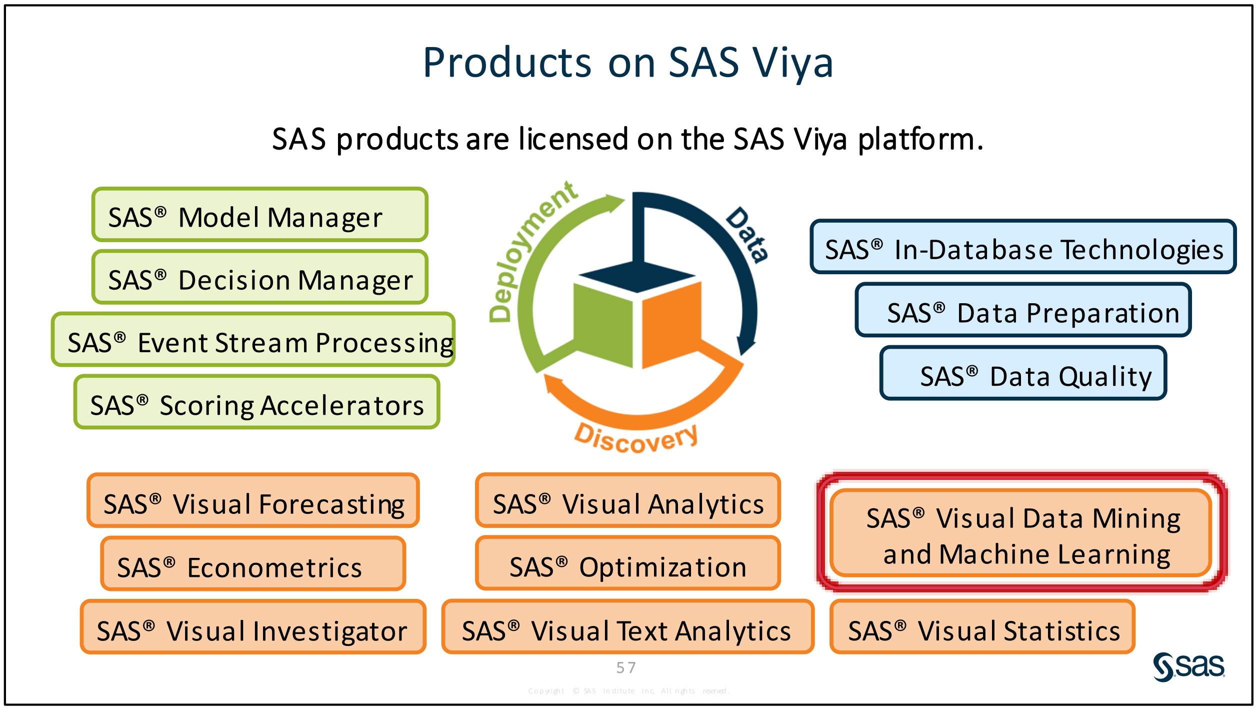
Although SAS Viya can be used by various SAS applications, it also enables you to access analytic methods from SAS, Python, Lua, and Java, as well as through a REST interface that uses HTTP or HTTPS.

APIs/UIs

- SAS
- Python
- REST
- Java
- R
- Lua
- SAS Studio

56

There are many interfaces to SAS Viya. From within each tool, you can extend your analysis into one of the others. Data can be shared, and models can be extended and compared.



Products on SAS Viya

SAS products are licensed on the SAS Viya platform.

SAS® Model Manager

SAS® Decision Manager

SAS® Event Stream Processing

SAS® Scoring Accelerators

Deployment

Data

Discovery

SAS® In-Database Technologies

SAS® Data Preparation

SAS® Data Quality

SAS® Visual Forecasting

SAS® Econometrics

SAS® Visual Investigator

SAS® Visual Analytics

SAS® Optimization

SAS® Visual Text Analytics

SAS® Visual Data Mining and Machine Learning

SAS® Visual Statistics

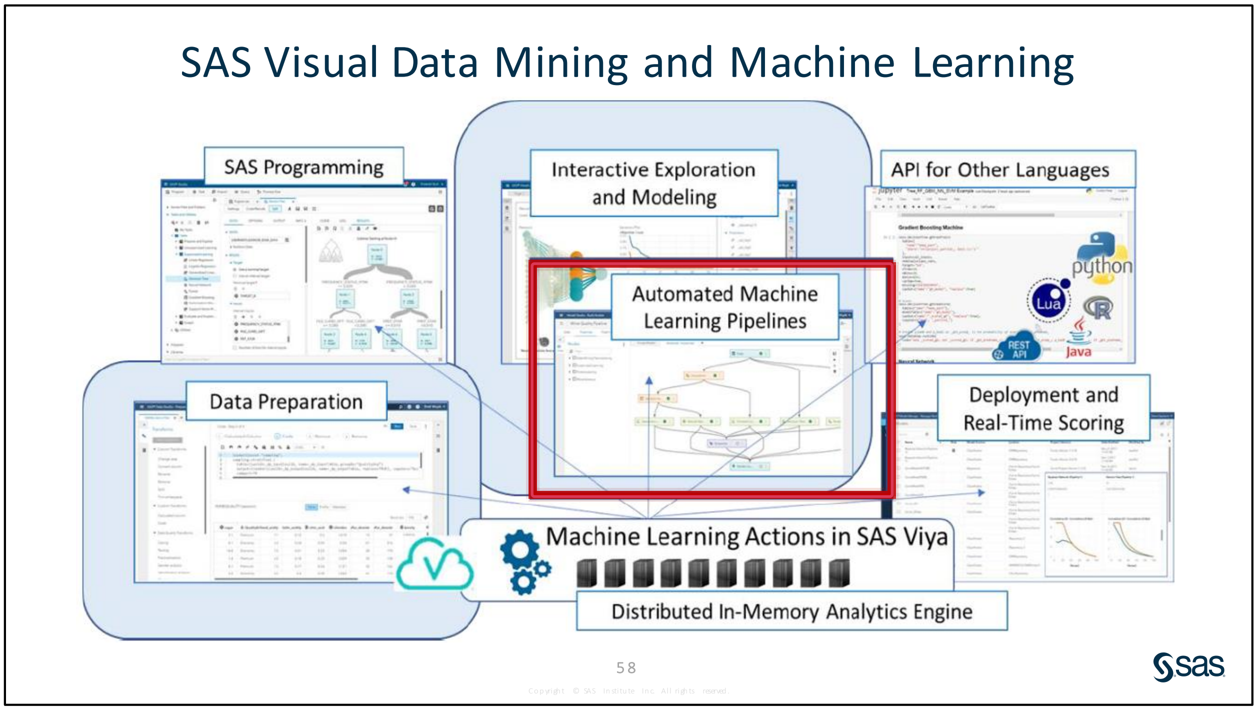
57

A variety of products sit on the SAS Viya platform. They enable users to perform their jobs as part of the analytics life cycle. In this course, you use SAS Visual Data Mining and Machine Learning, which provides end-to-end analytics.

SAS Viya includes following products:

SAS Visual Analytics	Visually explores all data, discovers new patterns, and publishes reports to the web and mobile devices.
SAS Visual Statistics	Adds an additional set of advanced analytic functionality that builds on SAS Visual Analytics.
SAS Visual Data Mining and Machine Learning	Surfaces in-memory machine-learning techniques such as gradient boosting, factorization machines, neural networks, and much more, through SAS Studio tasks, procedures, and a Python client. The visual interface is Model Studio, which provides integration between common analytical processes from data preparation, to exploration, to model development and deployment.
SAS Visual Text Analytics	Leverages powerful natural language processing, machine learning, and linguistic rules to reveal insights in data.
SAS Visual Forecasting	Leverages time series data to forecast future results.
SAS Econometrics	Provides techniques to model complex business and economic scenarios and analyze the dynamic impact that specific events might have over time.
SAS Optimization	Enables organizations to effectively consider more alternative actions and scenarios and determine the best allocation of resources and the best plans for accomplishing goals.
SAS Data Preparation	Quickly prepare data for analytics in a self-service, point-and-click environment with data preparation from SAS.
SAS Model Manager	Provides a framework for creating, managing, monitoring, and governing analytic models.
SAS Decision Manager	Streamlines analytical model and business rule deployment and automates operational business decisions.
SAS Event Stream Processing	Analyzes millions of events per second, detects patterns of interest as they occur, and decides what information should be acted on immediately, what can be ignored, and what should be stored.
SAS Visual Investigator	Analyzes and integrates disparate data throughout an organization and empowers users to uncover previously unknown relationships and insights.

Note: The Model Studio interface is superset of SAS Visual Data Mining and Machine Learning, SAS Visual Forecasting, and SAS Visual Text Analytics.



SAS Visual Data Mining and Machine Learning

SAS Programming

Interactive Exploration and Modeling

API for Other Languages

Automated Machine Learning Pipelines

Data Preparation

Deployment and Real-Time Scoring

Machine Learning Actions in SAS Viya

Distributed In-Memory Analytics Engine

58

SAS Visual Data Mining and Machine Learning is a product offering in SAS Viya that contains (1) underlying CAS actions and SAS procedures for data mining and machine learning applications, and (2) GUI-based applications for different levels and types of users. These applications are as follows:

- Programming interface: a collection of SAS procedures for direct coding or access through tasks in SAS Studio
- Interactive modeling interface: a collection of tasks in SAS Visual Analytics for creating models in an interactive manner with automated assessment visualizations
- Automated modeling interface: a pipeline application called Model Studio that enables you to construct automated flows consisting of various nodes for preprocessing and modeling, with automated model assessment and comparison, and direct model publishing and registration.

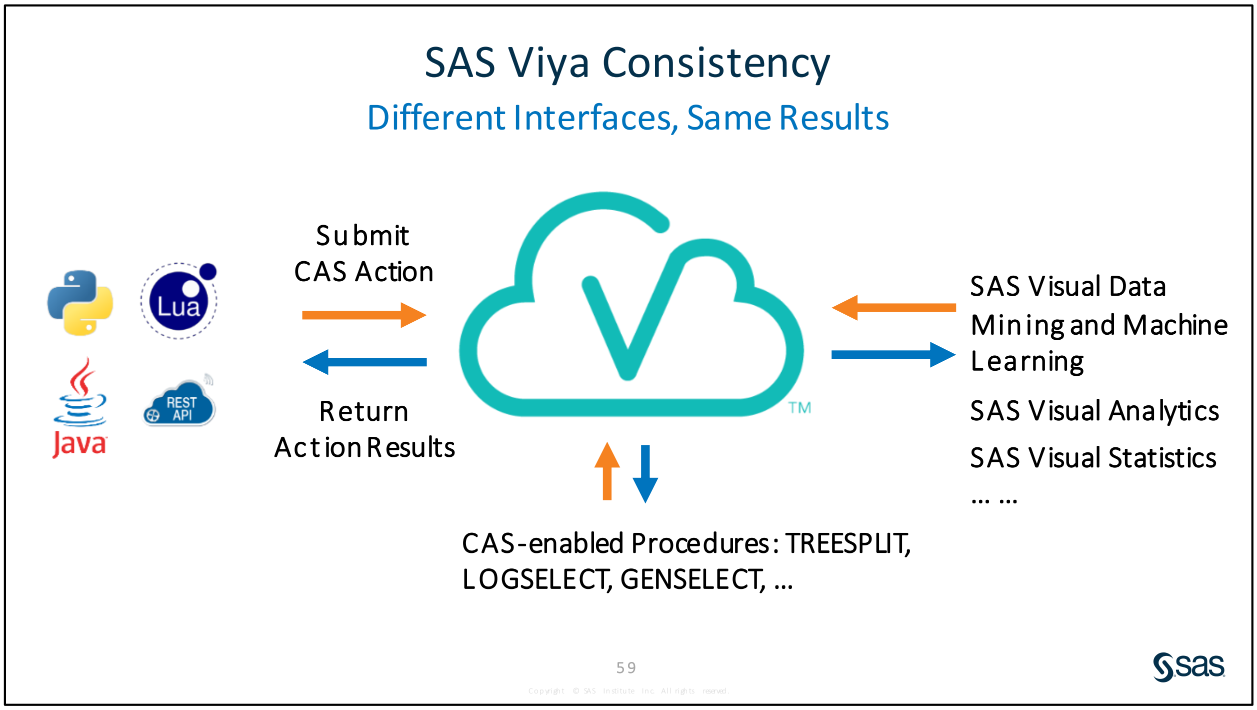
Each of these executes the same underlying actions in the CAS execution environment. In addition, there are supplementary interfaces for preparing your data (*SAS Data Studio*) and managing and deploying your models (*SAS Model Manager* and *SAS Decision Manager*) to support all phases of a machine learning application.

Note: In this course, you primarily explore the Model Studio interface and its integration with other SAS Visual Data Mining and Machine Learning interfaces.

You use the SAS Visual Data Mining and Machine Learning web client to visually assemble, configure, build, and compare data mining models and pipelines for a wide range of analytic data mining tasks. The software provides to new data miners a variety of end-to-end analytical modeling templates as well as the opportunity to create, modify, and save your own data mining tools, templates, and model score codes. SAS Visual Data Mining and Machine Learning provides support for your custom SAS code in the analytic pipeline models that you create.

The software expedites and simplifies model assessment and model pipeline comparisons when evaluating competing analytic models for the role of champion model. SAS Visual Data Mining and Machine Learning readily imports and exports data to other visual SAS analytic applications, as well as SAS Enterprise Miner on SAS 9.4 (and earlier) releases.

You can share projects and analyses developed in SAS Visual Data Mining and Machine Learning among concurrent users. SAS Visual Data Mining and Machine Learning generates APIs that enable model content and score code to be integrated with applications outside of SAS. SAS Visual Data Mining and Machine Learning supports integration with SAS Model Manager as well as many commercial databases.



SAS Viya Consistency

Different Interfaces, Same Results

SAS Visual Data Mining and Machine Learning

SAS Visual Analytics

SAS Visual Statistics

… …

CAS-enabled Procedures: TREESPLIT, LOGSELECT, GENSELECT, …

59

SAS Viya enables customers to develop, deploy, and manage enterprise-class analytical assets throughout the analytics life cycle with a single platform with the underlying engine called CAS, which is the acronym for Cloud Analytic Services.

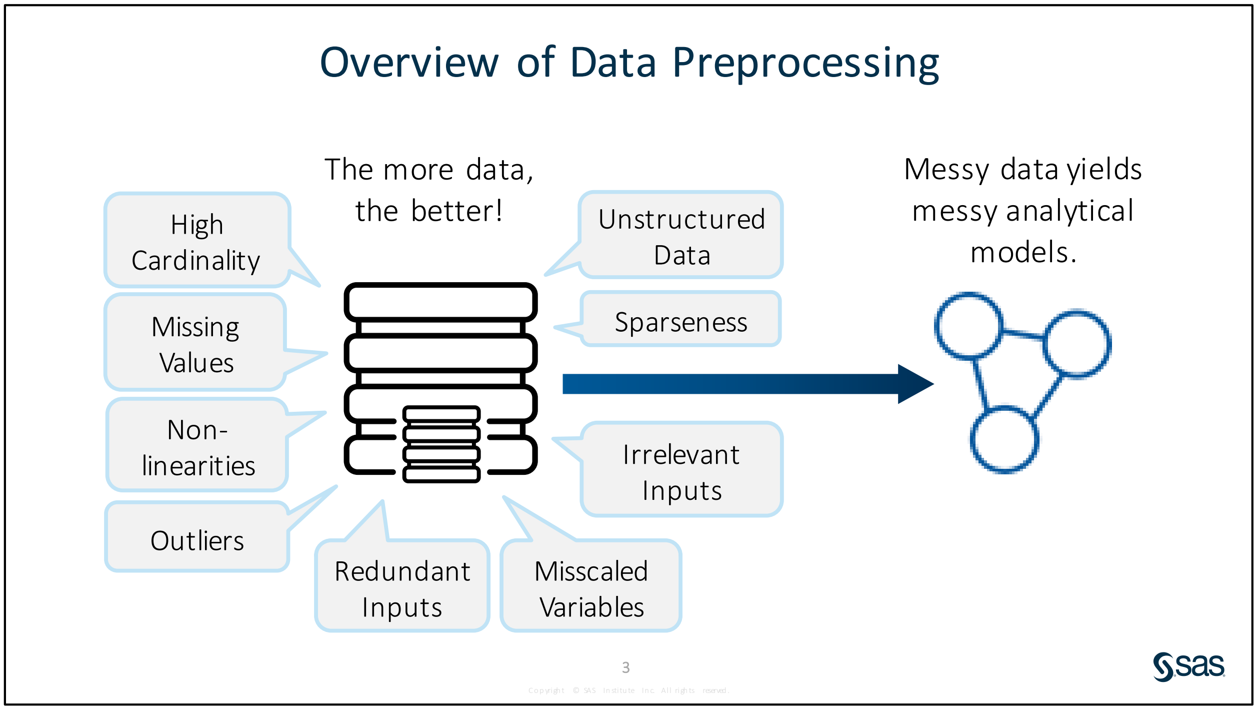
- SAS Viya delivers a single, consolidated, and centralized analytics environment. Customers no longer need to stitch together different analytic code bases.
- It natively supports programming in SAS and access to SAS from other languages such as R, Python, Java, and Lua. This means that data scientists and coders not familiar with SAS can use SAS Viya, but they do not need to learn SAS code.
- It supports access to SAS from third-party applications with public REST APIs, so developers can easily include SAS analytics in their applications.
- Regardless of which interface is used, the same CAS actions are applied behind the scenes for the same procedure. This provides important consistency.

Note: A CAS *action set* is a collection of actions that group functionality (for example, simple summary statistics). Many SAS procedures are being functionally converted into CAS actions and CAS action sets to be executed by the CAS server. Procedures that are executable by the CAS server and submitted from a SAS client can call the corresponding CAS actions. Then the actions produce output that is very similar to the SAS procedure.

A *CAS action*, the smallest unit of functionality in CAS, sends a request to the CAS server. The action parses the arguments of the request, invokes the action function, returns the results, and cleans the resources.

Chapter 2 Data Preparation

2.1 Data Exploration



Overview of Data Preprocessing

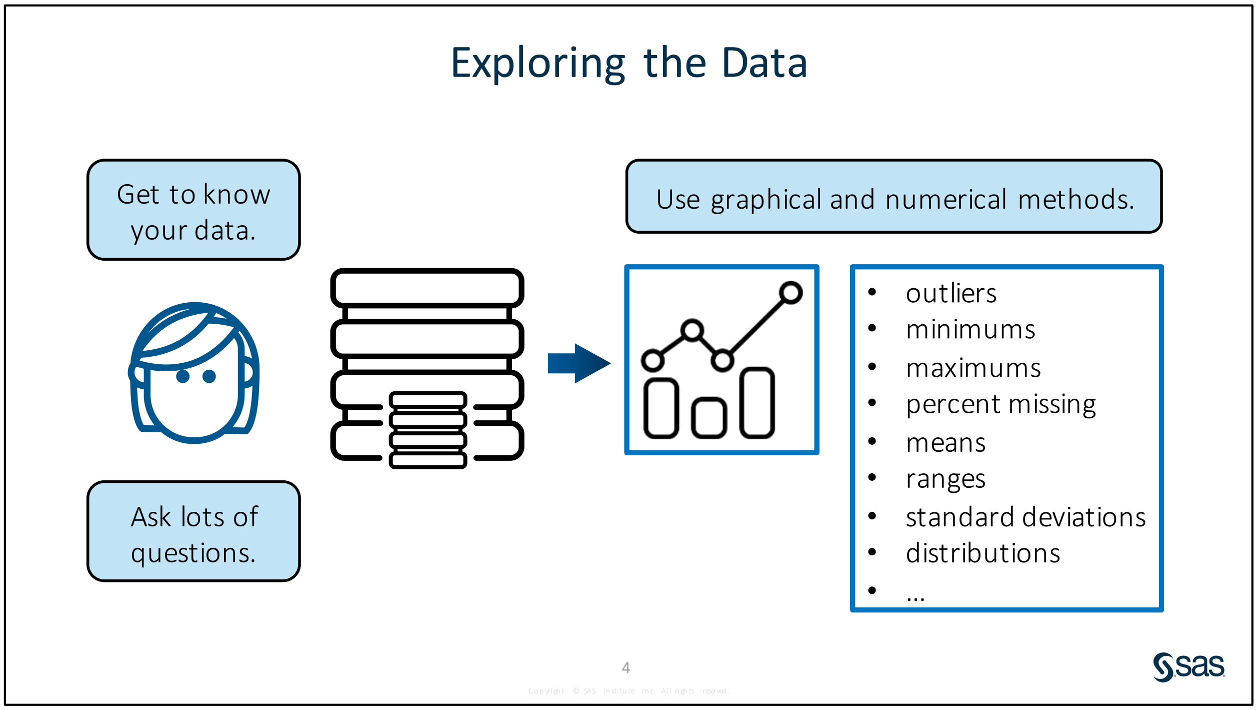
The more data, the better!

Messy data yields messy analytical models.

3

"The more data, the better" is the rule that generally prevails in any analytical exercise. However, real life data is usually dirty and noisy because of inconsistencies, incompleteness, duplication, and merging problems.

Preparing your data for analysis starts with exploring your data. Then, to clean up and reduce the data to a manageable and relevant size, you must apply various data preprocessing techniques. Remember the GIGO principle, which essentially states that messy data yields messy analytical models. Effective machine learning models are built on a foundation of well-prepared data. Before cleaning and transforming the data, you must think about how the data will be used. You must consider the analysis goal, the methods that you are using, and whether your data is appropriate in the first place. Shortcuts in data preprocessing hamper your models.



Exploring the Data

Get to know your data.

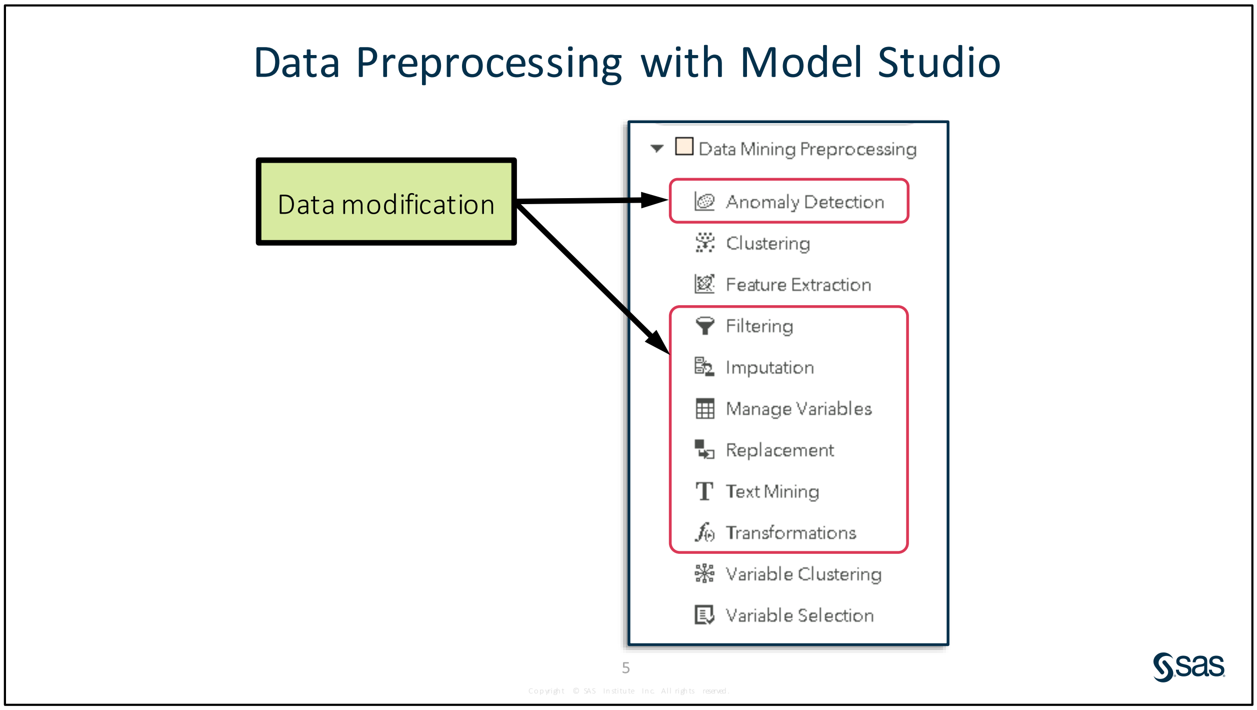
Ask lots of questions.

Use graphical and numerical methods.

- outliers
- minimums
- maximums
- percent missing
- means
- ranges
- standard deviations
- distributions
- ...

4

Exploring the data can be one of the most important and time-consuming parts of an analytical project. When exploring data, analysts try to gain intimate knowledge of the variables by using both graphical and numerical methods. Common graphical tools include histograms, scatter plots, bar charts, and stem-and-leaf plots. There are also more modern graphical tools, such as heat maps and word clouds, which scale well to large data sets. Numerical summary methods are also used to explore data. These include summary statistics for measure of central tendency such as the mean, median, or mode. Numeric measures of variability such as variance, standard deviation, range, or interquartile-range are also used to explore data. Extreme values such as outliers, the minimum, or the maximum are used to explore data as well as counts or percentages of missing data. Bivariate measures such as correlation are also used.



Data Preprocessing with Model Studio

5

Data preprocessing can occur in several places throughout SAS Visual Data Mining and Machine Learning: in a dedicated application (Data Studio), during visual exploration (SAS Visual Analytics), and during execution of a pipeline (Model Studio). Here we use the Model Studio application, which provides data preprocessing capabilities in the form of pipeline nodes. These nodes form a group called Data Mining Preprocessing.

Data modification is a broad preprocessing category. Any operation that alters the data or data roles can be considered as a modification, including dimension reduction techniques. Model Studio provides several SAS Visual Data Mining and Machine Learning nodes to modify your data.

Anomaly Detection

The Anomaly Detection node identifies and excludes anomalies using the support vector data description, or SVDD. Briefly, the SVDD formulation identifies outliers by determining the smallest possible hypersphere (built using support vectors) that encapsulates the training data points. The SVDD then excludes those data points that lie outside of the sphere that is built from the training data. Anomaly detection with SVDD is useful for data sets where most of the data belongs to one class and the other class is scarce or missing.

Filtering

The Filtering node excludes certain observations, such as rare values and outliers. Filtering extreme values from the training data tends to produce better models because the parameter estimates are more stable.

Imputation

The Imputation node replaces missing values in data sets. Simple imputation schemes include replacing a missing value in an input variable with the mean or mode of that variable's nonmissing values. For non-normally distributed variables or variables that have a high proportion of missing values, simple imputation might be ineffective. Imputation might also fail to be effective for variables whose missingness is not at random. For ideal results, create missing indicators and use them in the model alongside imputed variables. This practice can result in improved outcomes, even in cases where the variables are normally distributed and have few missing values.

Manage Variables

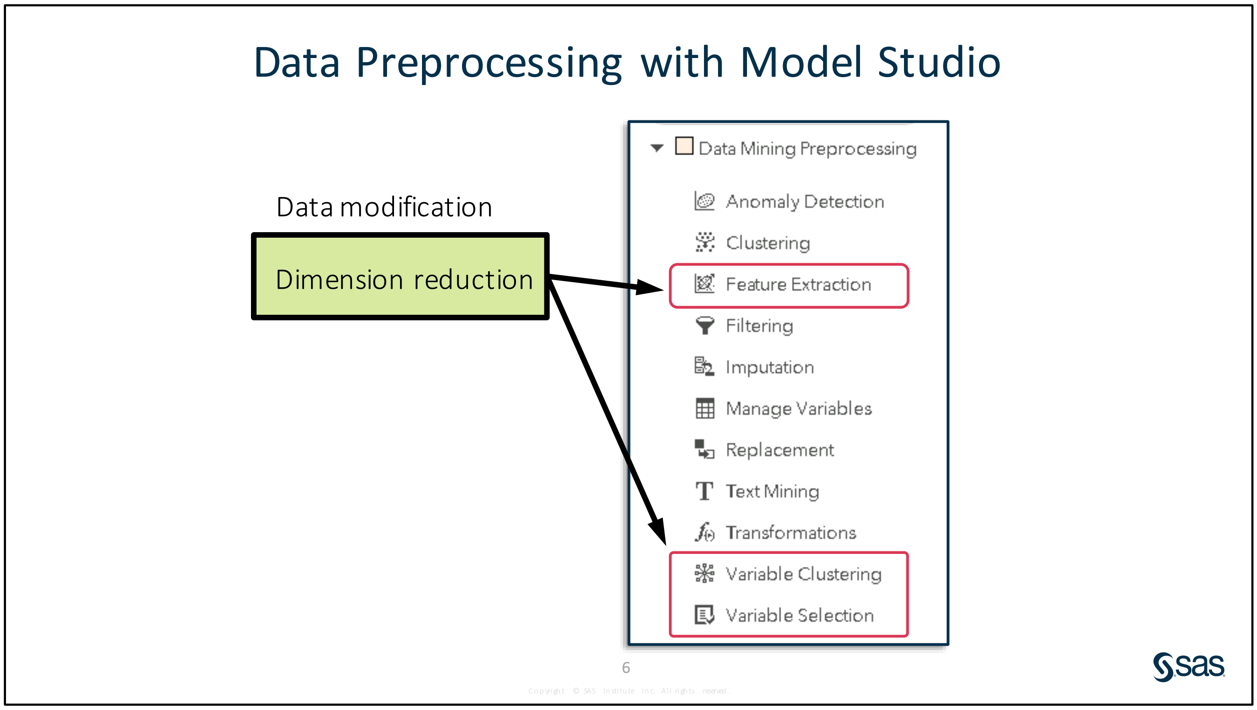
The Manage Variables node enables you to make modifications (such as changing the role of a variable or adding new transformations) to the data while within a Model Studio pipeline. The options available to you are a subset of the options available under the Data tab.

Replacement

The Replacement node enables you to replace outliers and unknown class levels with specified values. Much like with imputation, simple replacement of outliers and unknown class level is not always effective. Care should be taken to use replacement effectively.

Transformations

The Transformations node enables you to alter your data by replacing an input variable with some function of that variable. Transformations have many use cases. Transformations can be used to stabilize variances, remove nonlinearity, and correct non-normality.



Data Preprocessing with Model Studio

Data modification

Dimension reduction

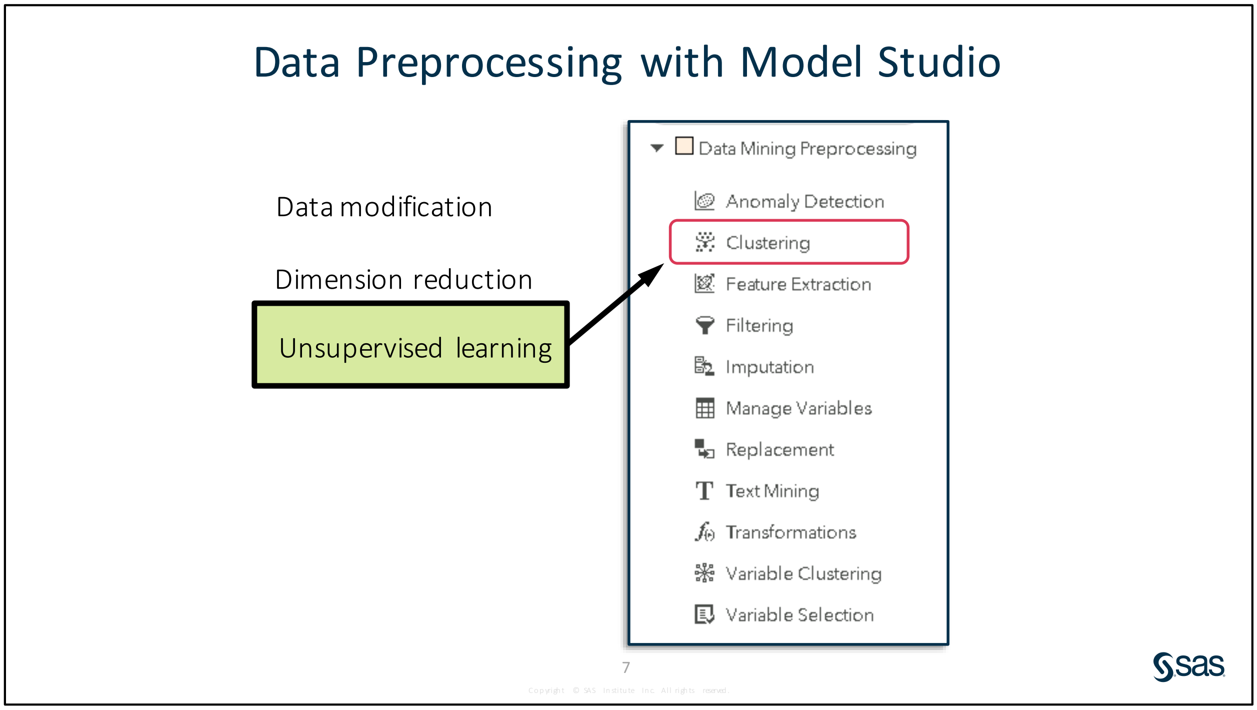
Dimension reduction decreases the number of variables under consideration. In many applications, the raw data has very high dimensional features, and some features are redundant or irrelevant to the task. Reducing the dimensionality helps find the true, latent relationship. Model Studio provides three nodes in SAS Visual Data Mining and Machine Learning for dimension reduction:

Feature Extraction

The Feature Extraction node transforms the existing features (variables) into a lower-dimensional space. Feature extraction in Model Studio is done using various techniques, including principal component analysis (PCA), robust PCA, singular value decomposition (SVD), and autoencoders. This is done by generating new features that are composites of the existing features. One drawback to feature extraction is that the composite variables are no longer meaningful with respect to the original problem.

Variable Clustering The Variable Clustering node divides numeric variables into disjoint clusters and chooses a variable that represents each cluster. Variable clustering removes collinearity, decreases redundancy, and helps reveal the underlying structure of the data set.

Variable Selection The Variable Selection node uses several unsupervised and supervised methods to determine which variables have the most impact on the model. Supervised variable selection techniques include variable selection based on linear models and tree-based models (such as decision tree, forest, and gradient boosting). This tool enables you to specify more than one selection technique, and there are several options for selection criteria. Because there can be disagreements on selected variables when different techniques are used, this functionality enables you to select variables that are consistently selected. Variables that fail to meet the selection criteria are marked as rejected and not used in successor modeling nodes.



Data Preprocessing with Model Studio

Data modification

Dimension reduction

Unsupervised learning

When performing *unsupervised learning*, the machine is presented with unlabeled data (unlabeled data has no target). Unsupervised learning algorithms seek to discover intrinsic patterns that underlie the data, such as a clustering or a redundant parameter (dimension) that can be reduced.

Model Studio provides the Clustering node to perform observation clustering based on distances that are computed from quantitative or qualitative variables (or both). The node uses the following algorithms:

- the *k*-means algorithm for clustering interval (quantitative) input variables
- the *k*-modes algorithm for clustering nominal (qualitative) input variables
- the *k*-prototypes algorithm for clustering mixed input that contains both interval and nominal variables.

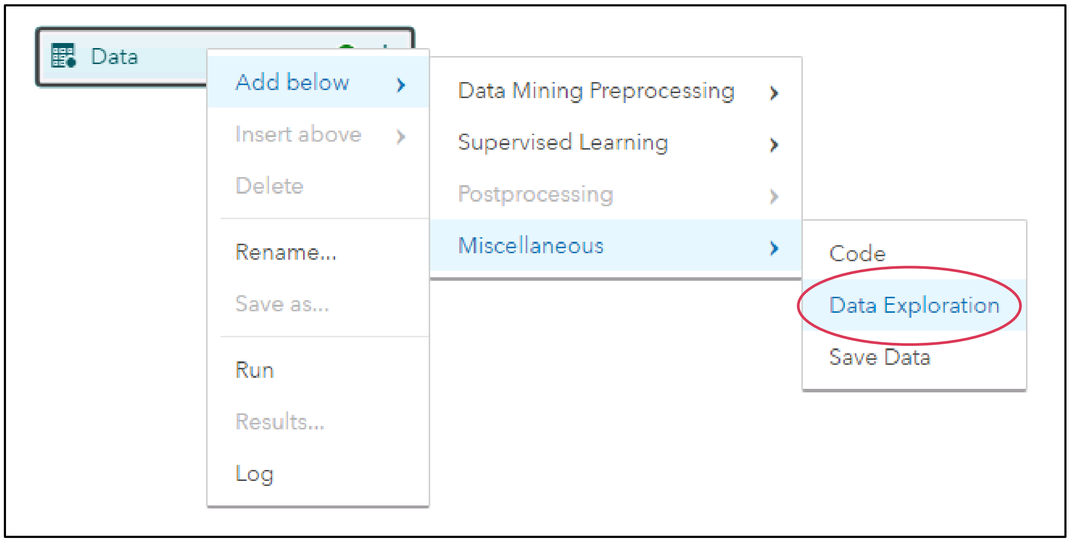
Clustering is often used to segment a large data set into several groups. Analysis can be performed in each group to help users find intrinsic patterns.

Exploring Source Data

In this demonstration, you use the Data Exploration node in Model Studio to assay and explore the **commsdata** data source. You will frequently find it useful to profile a data set before continuing your analysis. Here you select a subset of variables to provide a representative snapshot of the data. Variables can be selected to show the most important inputs, or to indicate suspicious variables (that is, variables with anomalous statistics).

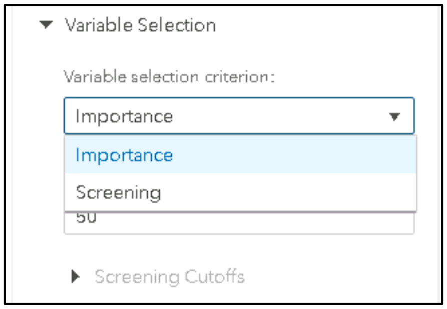
1. If you closed your browser and need to log back in, do the following:
   a. From the Windows taskbar, launch Google Chrome. When the browser opens, select **SAS Viya** ⇨ **SAS Home** from the bookmarks bar or from the link on the page.
   b. The **User ID** and **Password** fields should be pre-filled. If not, do the following:
      1) Enter **student** for the user ID.
      2) Enter **Metadata0** for the password.

      Note: Use caution when you enter the user ID and password because values are case sensitive.
   c. Click **Sign In**.
   d. Select **Yes** in the Assumable Groups window.
2. From the Model Studio Projects page, open the **Demo** project from the available existing projects.
3. Click the **Pipelines** tab. (You should be looking at the pipeline called **Pipeline 1**.)
4. Right-click the **Data** node and select **Add below** ⇨ **Miscellaneous** ⇨ **Data Exploration**.



Note: You can also drag the node from the left pane to the top of the Data node, and the node is added below.

5. Keep the default setting for **Variable selection criterion**, which is **Importance**. The variable selection criterion specifies whether to display the most important inputs or suspicious variables. The other possible value is **Screening**.



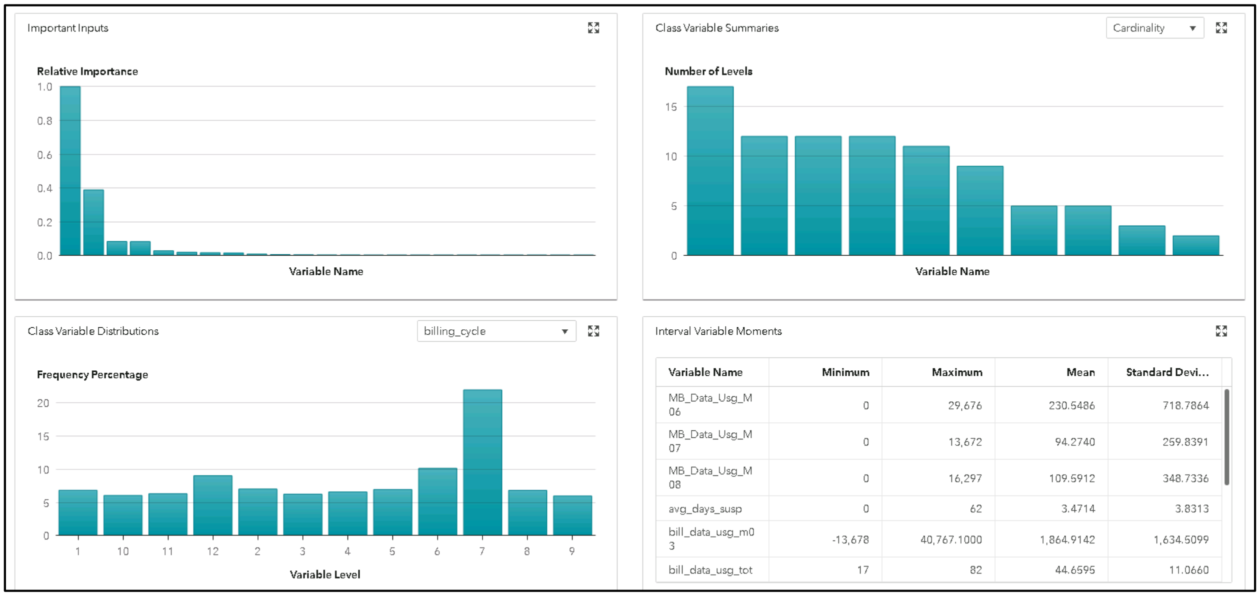
Variables can be selected to show the most important inputs or to indicate anomalous statistics. By default, a maximum of 50 variables will be selected with the **Importance** criterion.

You can control the selection of suspicious variables by specifying screening criteria like cutoff for flagging variables with a high percentage of missing values, high cardinality class variables, class variables with dominant levels, class variables with rare modes, skewed interval variables, peaky (leptokurtic) interval variables, and interval variables with thick tails (that is, platykurtic distributions).

6. Right-click the **Data Exploration** node and select **Run** (or click the right-pointing arrow ▶ icon).

The Data Exploration node gives you a statistical summary of the input data. This node can be a useful first step in analysis because it provides a subset of variables that are a representative snapshot of the data. The Data Exploration node can be placed most anywhere in a pipeline except after the Model Comparison node.

7. When the pipeline finishes running, right-click the **Data Exploration** node and select **Results**.



Interval Variable Moments

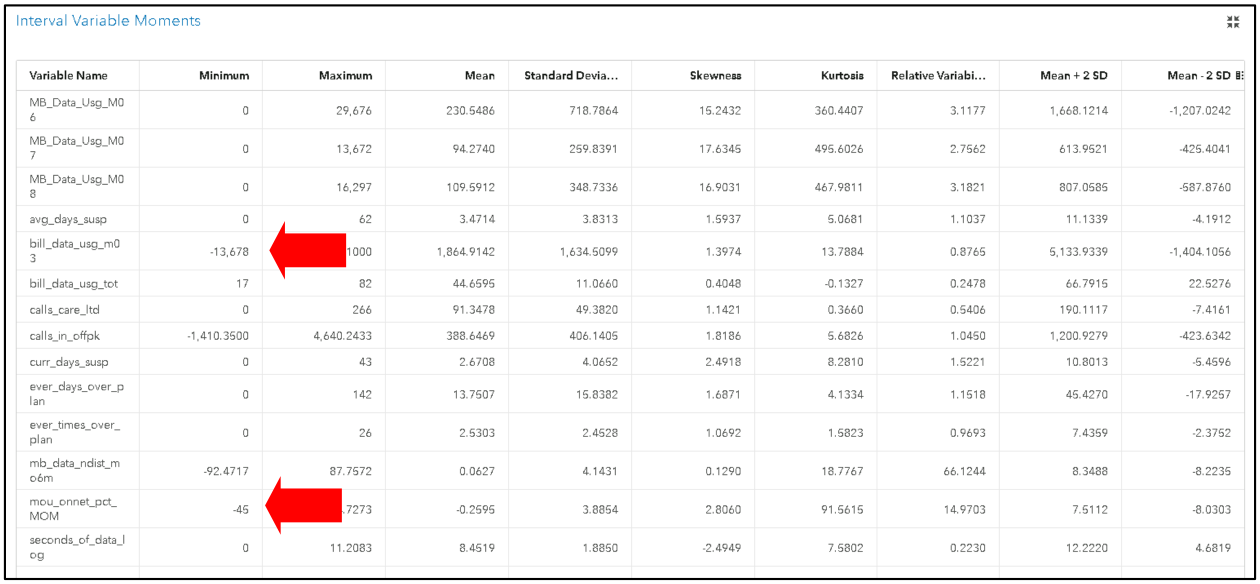
Variable Name	Minimum	Maximum	Mean	Standard Devi...
MB_Data_Usg_M06	0	29,676	230.5486	718.7864
MB_Data_Usg_M07	0	13,672	94.2740	259.8391
MB_Data_Usg_M08	0	16,297	109.5912	348.7336
avg_days_susp	0	62	3.4714	3.8313
bill_data_usg_m03	-13,678	40,767.1000	1,864.9142	1,634.5099
bill_data_usg_tot	17	82	44.6595	11.0660

8. Click to expand the **Importance Inputs** bar chart and examine the relative importance of the ranked variables. This bar chart is available only if **Variable selection criterion** is set to **Importance**.

   The relative variable importance metric is a number between 0 and 1, which is calculated in two steps. First, it finds the maximum RSS-based variable importance. This method measures variable importance based on the change of residual sum of squares (RSS) when a split is found at a node. Second, for each variable, it calculates the relative variable importance as the RSS-based importance of this variable divided by the maximum RSS-based importance among all the variables. The RSS and relative importance are calculated from the validation data. If no validation data exist, these two statistics are calculated instead from the training data.

9. Click to exit the maximized view.

10. Expand the **Interval Variable Moments** table.



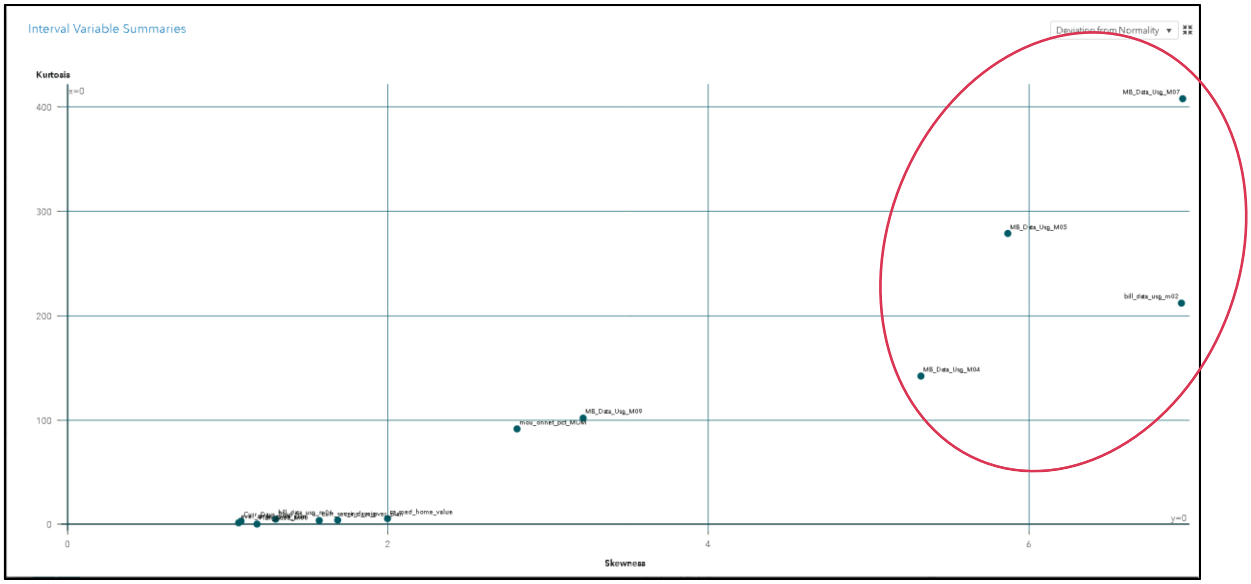
Interval Variable Moments

Variable Name	Minimum	Maximum	Mean	Standard Devia...	Skewness	Kurtosis	Relative Variabi...	Mean + 2 SD	Mean - 2 SD
MB_Data_Usg_M06	0	29,676	230.5486	718.7864	15.2432	360.4407	3.1177	1,668.1214	-1,207.0242
MB_Data_Usg_M07	0	13,672	94.2740	259.8391	17.6345	495.6026	2.7562	613.9521	-425.4041
MB_Data_Usg_M08	0	16,297	109.5912	348.7336	16.9031	467.9811	3.1821	807.0585	-587.8760
avg_days_susp	0	62	3.4714	3.8313	1.5937	5.0681	1.1037	11.1339	-4.1912
bill_data_usg_m03	-13,678	1000	1,864.9142	1,634.5099	1.3974	13.7884	0.8765	5,133.9339	-1,404.1056
bill_data_usg_tot	17	82	44.6595	11.0660	0.4048	-0.1327	0.2478	66.7915	22.5276
calls_care_ltd	0	266	91.3478	49.3820	1.1421	0.3660	0.5406	190.1117	-7.4161
calls_in_offpk	-1,410.3500	4,640.2433	388.6469	406.1405	1.8186	5.6826	1.0450	1,200.9279	-423.6342
curr_days_susp	0	43	2.6708	4.0652	2.4918	8.2810	1.5221	10.8013	-5.4596
ever_days_over_plan	0	142	13.7507	15.8382	1.6871	4.1334	1.1518	45.4270	-17.9257
ever_times_over_plan	0	26	2.5303	2.4528	1.0692	1.5823	0.9693	7.4359	-2.3752
mb_data_ndist_mo6m	-92.4717	87.7572	0.0627	4.1431	0.1290	18.7767	66.1244	8.3488	-8.2235
mou_onnet_pct_MOM	-45	.7273	-0.2595	3.8854	2.8060	91.5615	14.9703	7.5112	-8.0303
seconds_of_data_log	0	11.2083	8.4519	1.8850	-2.4949	7.5802	0.2230	12.2220	4.6819

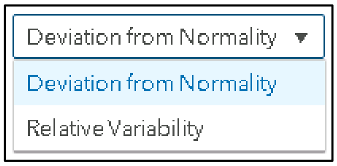
   This table displays the interval variables with their associated statistics, which include minimum, maximum, mean, standard deviation, skewness, kurtosis, relative variability, and the mean plus or minus two standard deviations. Note that some of the input variables have negative values.

11. Click to exit the maximized view of this window.

12. Scroll down in the Data Exploration Results window to examine the Interval Variable Summaries scatter plot. Observe that several variables have deviation from normality—that is, high kurtosis on the Y axis and high skewness on the X axis.



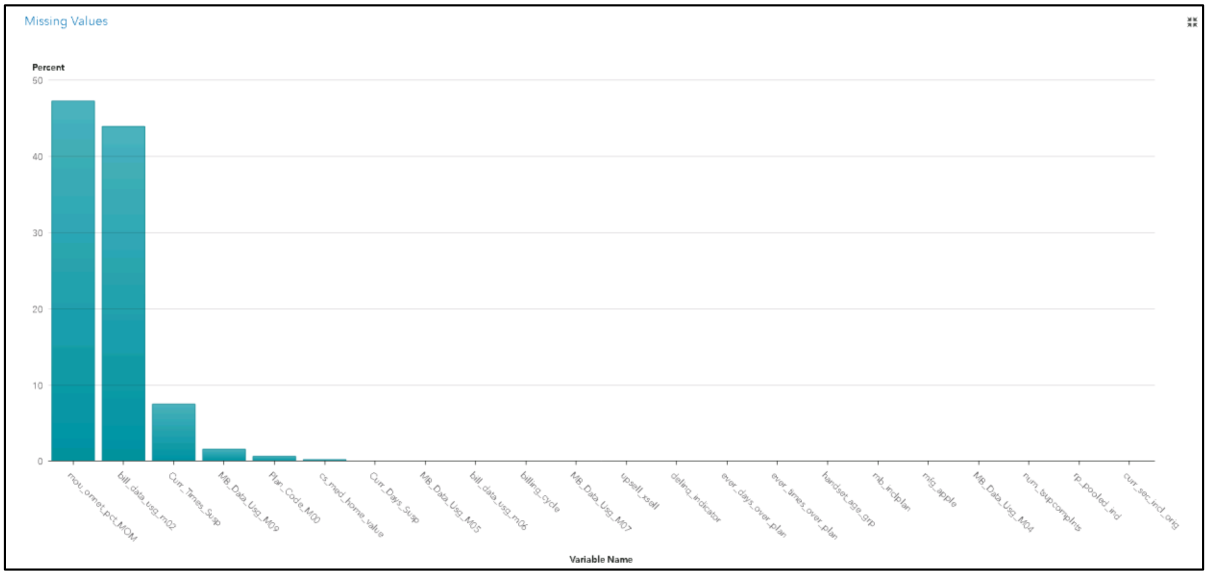
13. Use the drop-down menu in the upper right corner to examine a bar chart of the relative variability for each interval variable.



14. Click to exit the maximized view of this window.

Note: Relative variability is useful for comparing variables with similar scales, such as several income variables. Relative Variability is the coefficient of variation, which is a measure of variance relative to the mean, CV=σ/μ.

15. Scroll down in the Data Exploration Results window to examine the Missing Values bar chart and validate that quite a few variables have missing values.



This is an important finding that you address in the next demonstration.

16. Click **Close**.

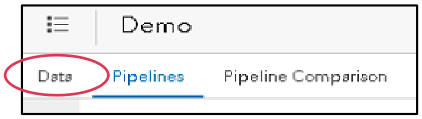
17. Double-click the **Pipeline 1** tab and rename it **Data Exploration**. Press the Enter key.

End of Demonstration

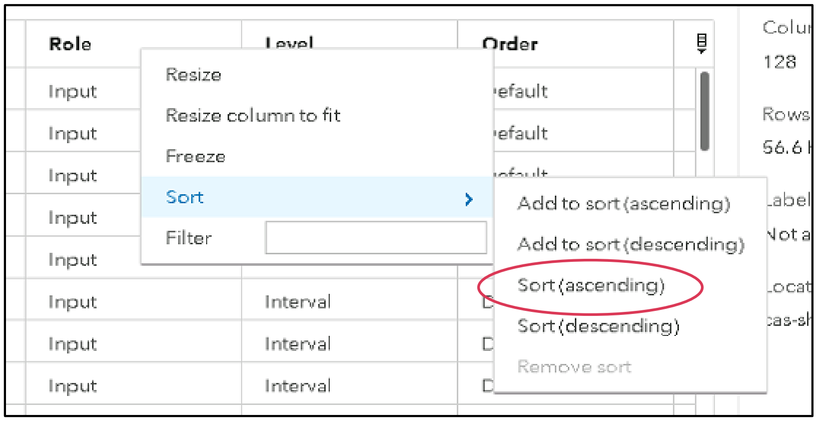
Modifying and Correcting Source Data Using the Data Tab

In this demonstration, you use the Data tab and Replacement node to modify a data source.

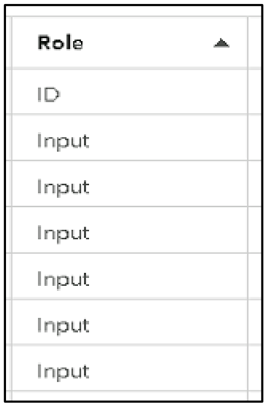
1. Click the **Data** tab.



2. Right-click the **Role** column and select **Sort** ⇨ **Sort (ascending)**.

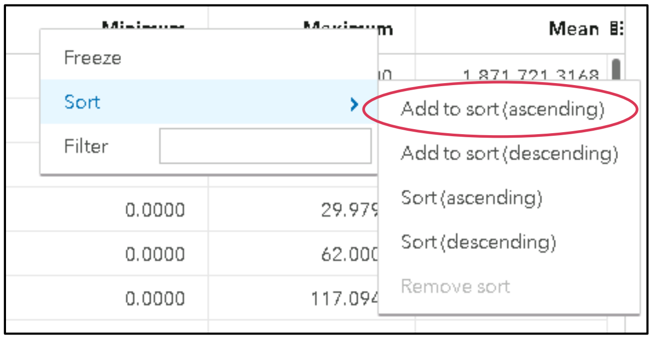


All the Input variables are grouped together after the ID variable(s) and before the Rejected variables.



3. Scroll to the right. Right-click the **Minimum** column and select **Sort** ⇨ **Add to sort (ascending)**.

   Variables with negative minimum values are grouped together.



   Note: **Add to sort** means that the initial sorting done by role still holds, so the sort on minimum values takes place within each sorted role group.

4. Select the following ***interval*** input variables:

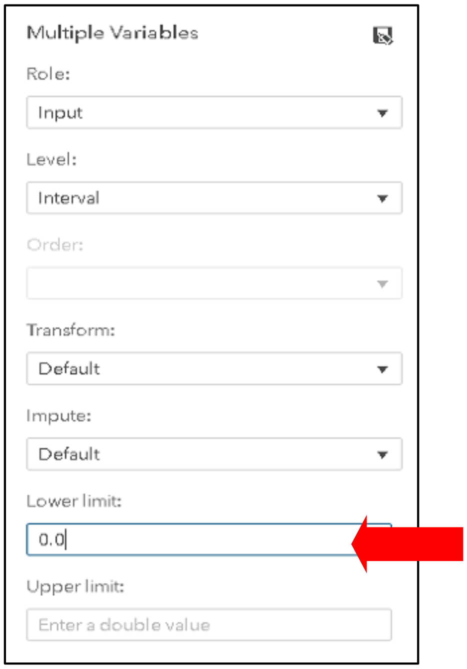
   - **tot_mb_data_roam_curr**
   - **seconds_of_data_norm**
   - **lifetime_value**
   - **bill_data_usg_m03**
   - **bill_data_usg_m06**
   - **voice_tot_bill_mou_curr**
   - **tot_mb_data_curr**
   - **mb_data_usg_roamm01 - mb_data_usg_roamm03**
   - **mb_data_usg_m01 - mb_data_usg_m03**
   - **calls_total**
   - **call_in_pk**
   - **calls_out_pk**
   - **call_in_offpk**
   - **calls_out_offpk**
   - **mb_data_ndist_mo6m**
   - **data_device_age**
   - **mou_onnet_pct_MOM**
   - **mou_total_pct_MOM**

   (Scroll back to the left to find the Variable Name column. You select 22 interval input variables.)

   Note: Selecting the check box of a variable and then selecting another variable while holding down the Shift key selects those two variables and all the variables between them.

5. On the right pane, enter **0.0** in the **Lower Limit** field in the Multiple Variables window. This specifies the lower limit to be used in the Filtering and Replacement nodes with the Metadata limits method.

   Note: This is a customer billing data, and negative values often imply that there is a credit applied to the customer's account. So it is not outside the realm of possibility that there are negative numbers in these columns. However, there is a general practice to convert negative values to zeros in telecom data.



Multiple Variables

Role: Input

Level: Interval

Order:

Transform: Default

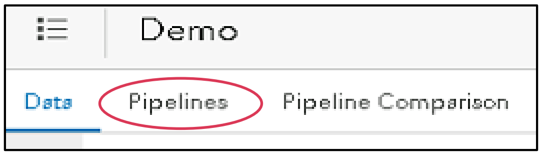
Impute: Default

Lower limit: 0.0

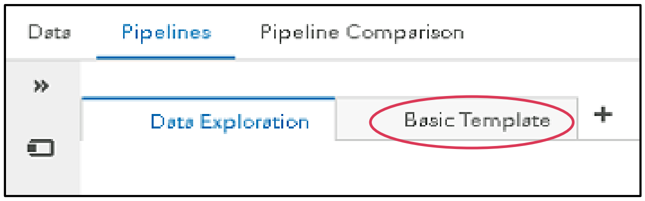
Upper limit: Enter a double value

   Note that you did not edit any variable values. Instead, you have just set a metadata property that can be invoked.

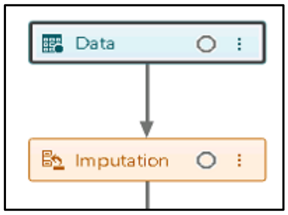
6. Click the **Pipelines** tab.



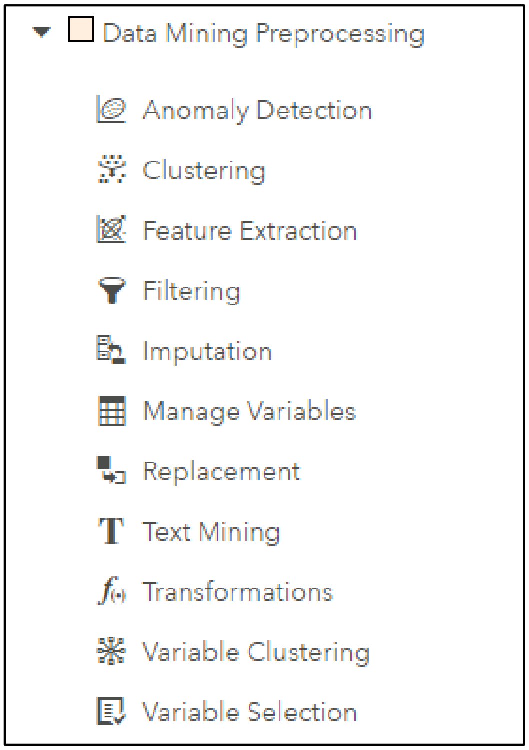
7. Select the **Basic Template** pipeline.



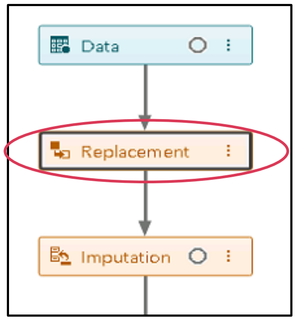
Notice that because of the change in metadata, the green check marks in the nodes in the pipeline have been changed to gray circles. This indicates that the nodes need to be rerun to reflect the change in metadata. The nodes will show the green check marks again when the pipeline is rerun.



8. Expand the **Nodes** pane on the left side of the canvas.
9. Expand **Data Mining Preprocessing**.

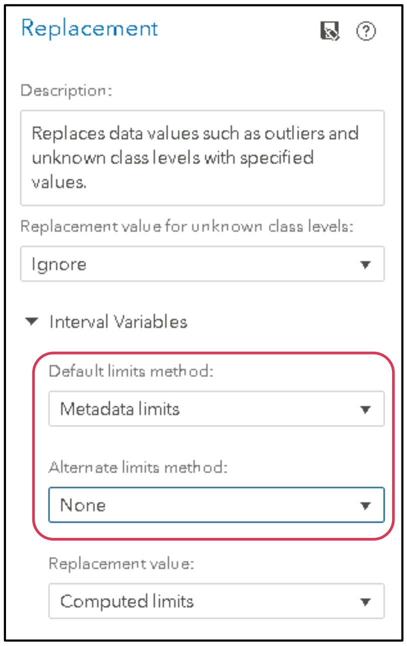


10. Click and drag the **Replacement** node and drop it ***between*** the Data node and the Imputation node.



The Replacement node can be used to replace outliers and unknown class levels with specified values. This is where you invoke the metadata property of the lower limit that you set before.

11. In the options panel on the right side, complete the following for the Interval Variables section:
    a. Set **Default limits method** to **Metadata limits**.
    b. Change **Alternate limits method** to **None**.
    c. Leave **Replacement value** as the default, **Computed limits**.



Replacement

Description:

Replaces data values such as outliers and unknown class levels with specified values.

Replacement value for unknown class levels: Ignore

Interval Variables

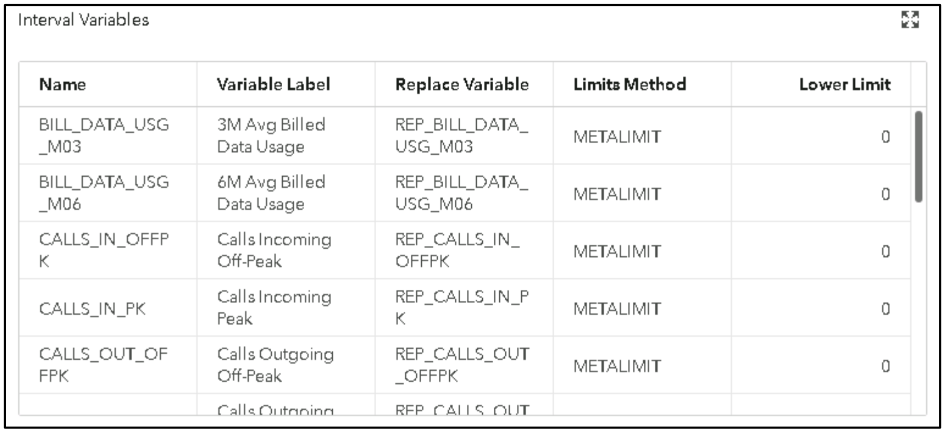
Default limits method: Metadata limits

Alternate limits method: None

Replacement value: Computed limits

12. Right-click the **Replacement** node and select **Run**. Negative values are replaced with zeros in the training partition of the data.

13. View the results of the Replacement node. The Interval Variables table shows which variables now have a lower limit of 0.



Interval Variables

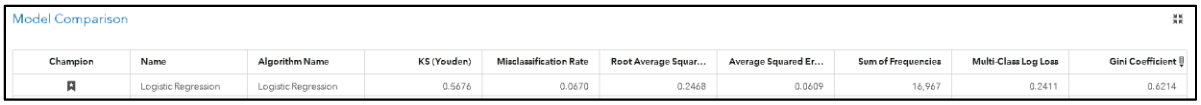
Name	Variable Label	Replace Variable	Limits Method	Lower Limit
BILL_DATA_USG_M03	3M Avg Billed Data Usage	REP_BILL_DATA_USG_M03	METALIMIT	0
BILL_DATA_USG_M06	6M Avg Billed Data Usage	REP_BILL_DATA_USG_M06	METALIMIT	0
CALLS_IN_OFFPK	Calls Incoming Off-Peak	REP_CALLS_IN_OFFPK	METALIMIT	0
CALLS_IN_PK	Calls Incoming Peak	REP_CALLS_IN_PK	METALIMIT	0
CALLS_OUT_OFFPK	Calls Outgoing Off-Peak	REP_CALLS_OUT_OFFPK	METALIMIT	0
	Calls Outgoing	REP_CALLS_OUT		

14. Close the results window of the Replacement node.

15. To update the remainder of the results of the pipeline, click the right-pointing arrow ▶ icon.

16. Right-click the **Model Comparison** node and select **Results**.

17. Click ⛶ to expand the Model Comparison table. Click ⛶ to exit the maximized view.



Model Comparison

Champion	Name	Algorithm Name	KS (Youden)	Misclassification Rate	Root Average Squar...	Average Squared Er...	Sum of Frequencies	Multi-Class Log Loss	Gini Coefficient
🔖	Logistic Regression	Logistic Regression	0.5676	0.0670	0.2468	0.0609	16,967	0.2411	0.6214

18. Select **Close** to return to the pipeline.
19. Double-click the **Basic Template** tab and rename it **Starter Template**. Press the Enter key.

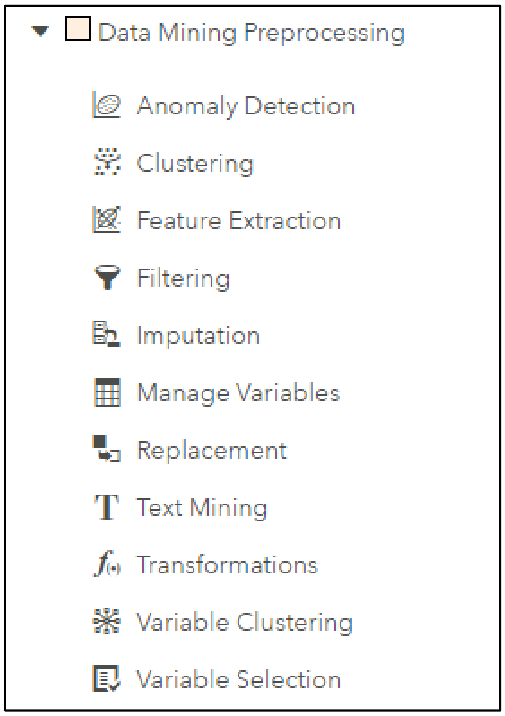
End of Demonstration

Alternate Method to Modify and Correct Source Data Using the Manage Variables Node (Self-Study)

In addition to using the Data tab, the Manage Variables node is another powerful tool that can be used to modify and correct data. The Manage Variables node is used directly within a pipeline.

Note: On the classroom computers, the Data tab should be used to modify and correct source data as described in the previous demonstration. That method is needed for other classroom needs. This demonstration provides another means of assigning metadata rules to data. ***One drawback to the method shown in this demonstration is that rules defined in the Manage Variables node are not saved if the pipeline is saved to the Exchange.***

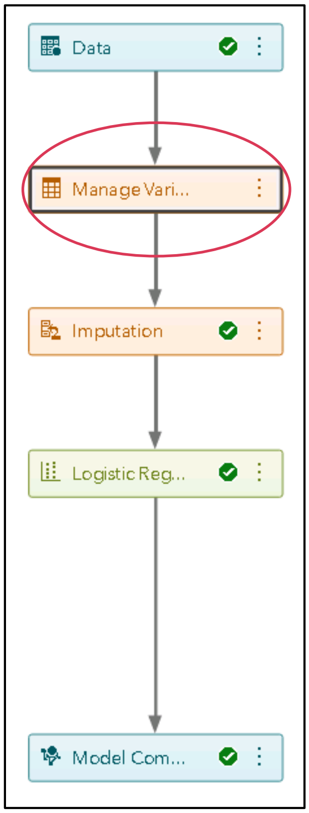
1. Select the **Basic Template** pipeline.
2. Expand the **Nodes** pane on the left side of the canvas.
3. Expand **Data Mining Preprocessing**.



▼ Data Mining Preprocessing

- Anomaly Detection
- Clustering
- Feature Extraction
- Filtering
- Imputation
- Manage Variables
- Replacement
- Text Mining
- Transformations
- Variable Clustering
- Variable Selection

4. Click and drag the **Manage Variables** node and drop it ***between*** the Data node and the Imputation node.

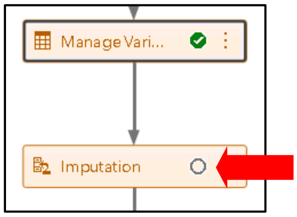


The Manage Variables node is a preprocessing node that enables you to make modifications to the metadata while it is within a Model Studio pipeline.

5. Right-click the **Manage Variables** node and select **Run**.

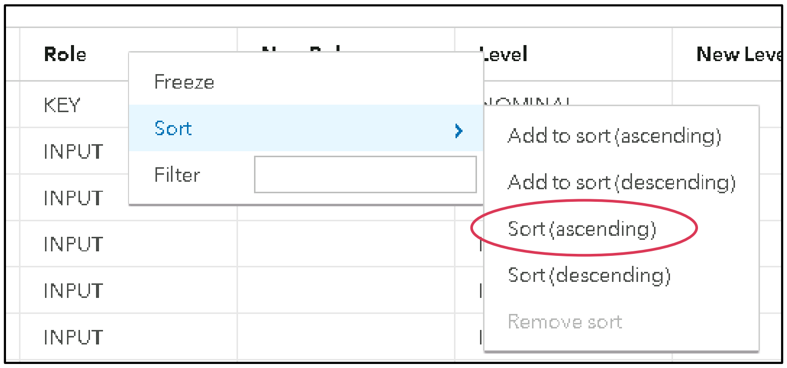
This reads the observations and variables and sets up the incoming variables before modifying their metadata.

Note: Notice that after the Manage Variables node is run, any nodes beneath it in the same path change in appearance from showing a green check mark to showing a gray circle.

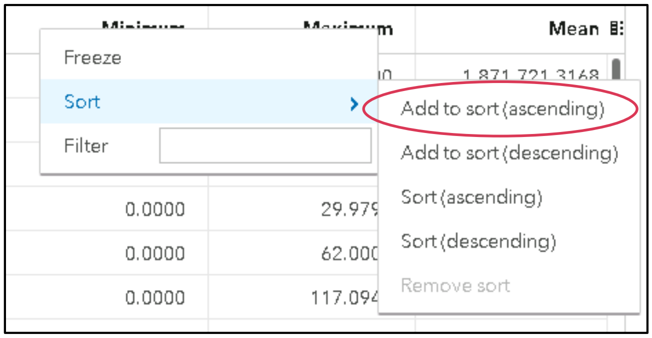


6. After the node runs, right-click the node again and select the **Manage Variables** option.

7. In the Manage Variables window, right-click the **Role** column and select **Sort** ⇨ **Sort (ascending)**.



8. Scroll to the right. Right-click the **Minimum** column and select **Sort** ⇨ **Add to sort (ascending)**.



9. Modify the following properties of the variables specified:

   a. To modify the metadata of a variable in the Manage Variables window, first select the desired variable. (You might want to select several variables at one time.)

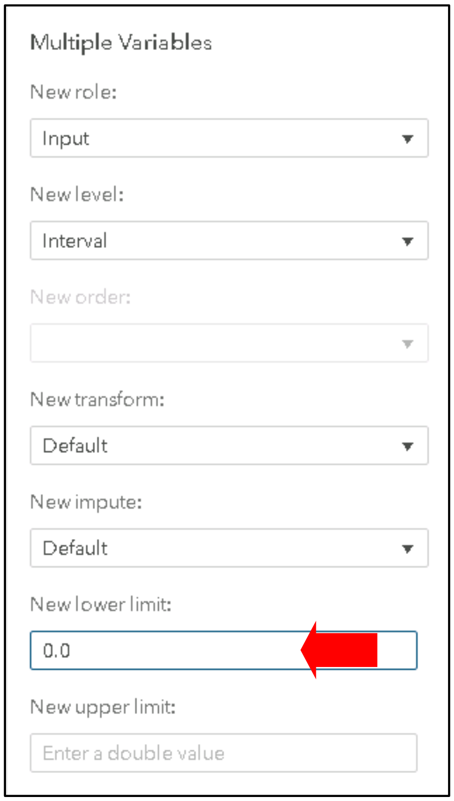
   Select the following ***interval*** input variables:

   - **tot_mb_data_roam_curr**
   - **seconds_of_data_norm**
   - **lifetime_value**
   - **bill_data_usg_m03**
   - **bill_data_usg_m06**
   - **voice_tot_bill_mou_curr**
   - **tot_mb_data_curr**
   - **mb_data_usg_roamm01 - mb_data_usg_roamm03**
   - **mb_data_usg_m01 - mb_data_usg_m03**
   - **calls_total**
   - **call_in_pk**
   - **calls_out_pk**
   - **call_in_offpk**
   - **calls_out_offpk**
   - **mb_data_ndist_mo6m**
   - **data_device_age**

- **mou_onnet_pct_MOM**
- **mou_total_pct_MOM**

(Scroll back to the left to find the Variable Name column. You select 22 interval input variables.)

b. Next, in the right pane, enter **0.0** into the **New Lower Limit** field in the Multiple Variables window.



Multiple Variables

New role: Input

New level: Interval

New order:

New transform: Default

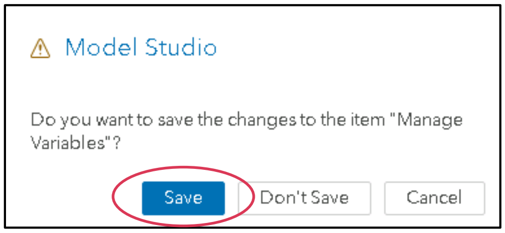
New impute: Default

New lower limit: 0.0

New upper limit: Enter a double value

No variable values were edited. Instead, a metadata property has been defined that can be invoked by using an appropriate node.

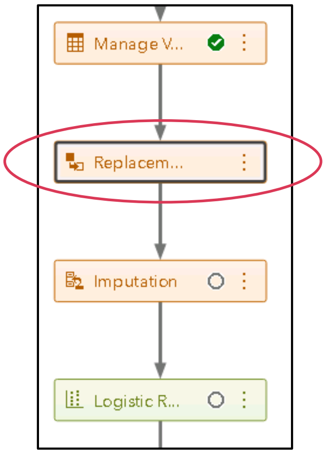
10. Select **Close** to exit the Manage Variables window. Click **Save** when asked if you want to save the changes. The attribute alterations are applied.



Model Studio

Do you want to save the changes to the item "Manage Variables"?

11. Click and drag the **Replacement** node and drop it ***between*** the Manage Variables node and the Imputation node.

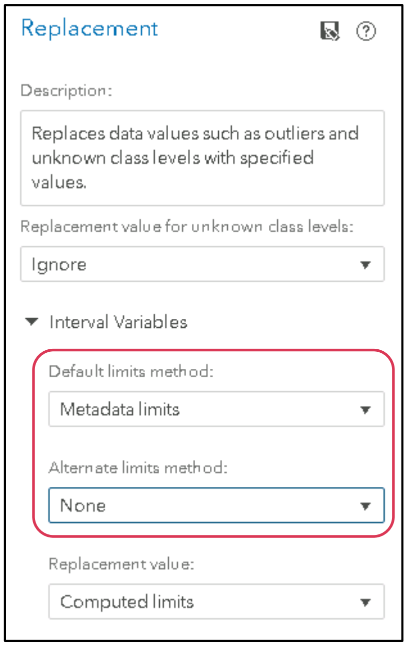


12. In the options panel on the right side, complete the following for Interval Variables:

    a. Set **Default limits method** to **Metadata limits**.

    b. Change **Alternate limits method** to **None**.

    c. Leave **Replacement value** as the default, **Computed limits**.



Replacement

Description:

Replaces data values such as outliers and unknown class levels with specified values.

Replacement value for unknown class levels: Ignore

Interval Variables

Default limits method: Metadata limits

Alternate limits method: None

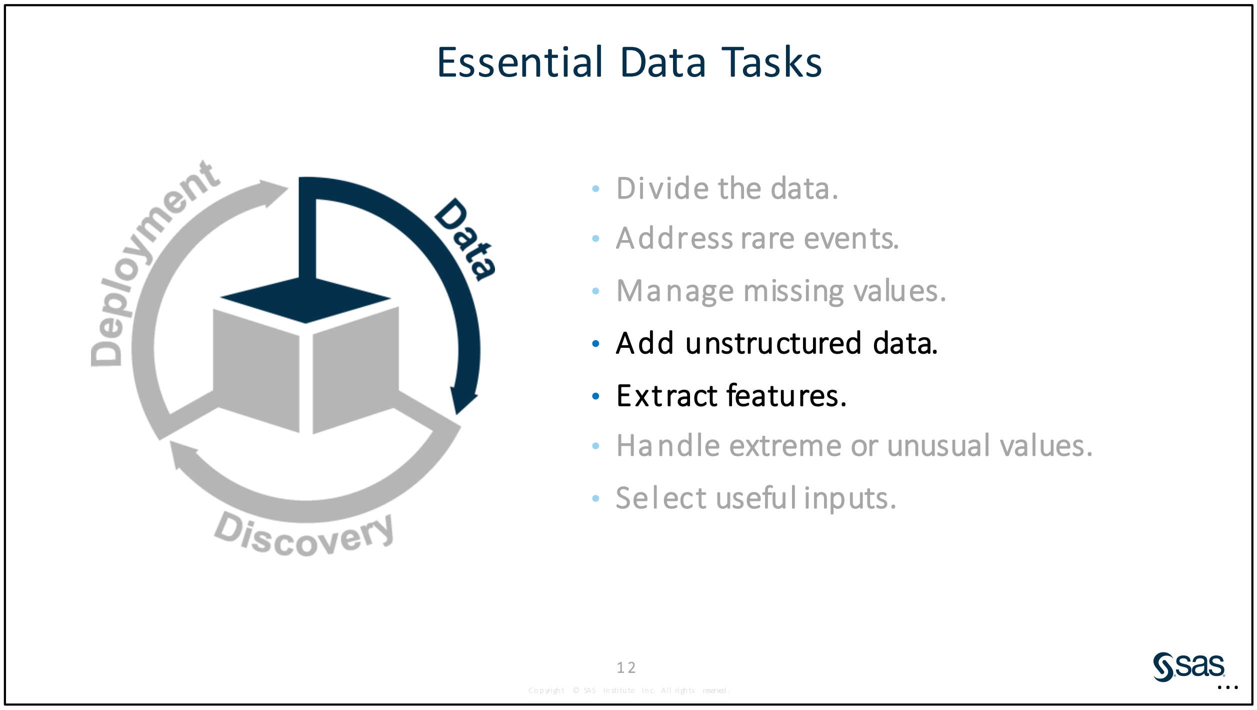
Replacement value: Computed limits

13. Right-click the **Replacement** node and select **Run**. Negative values are replaced with zeros in the training partition of the data.

14. Close the results window of the Replacement node.

End of Demonstration

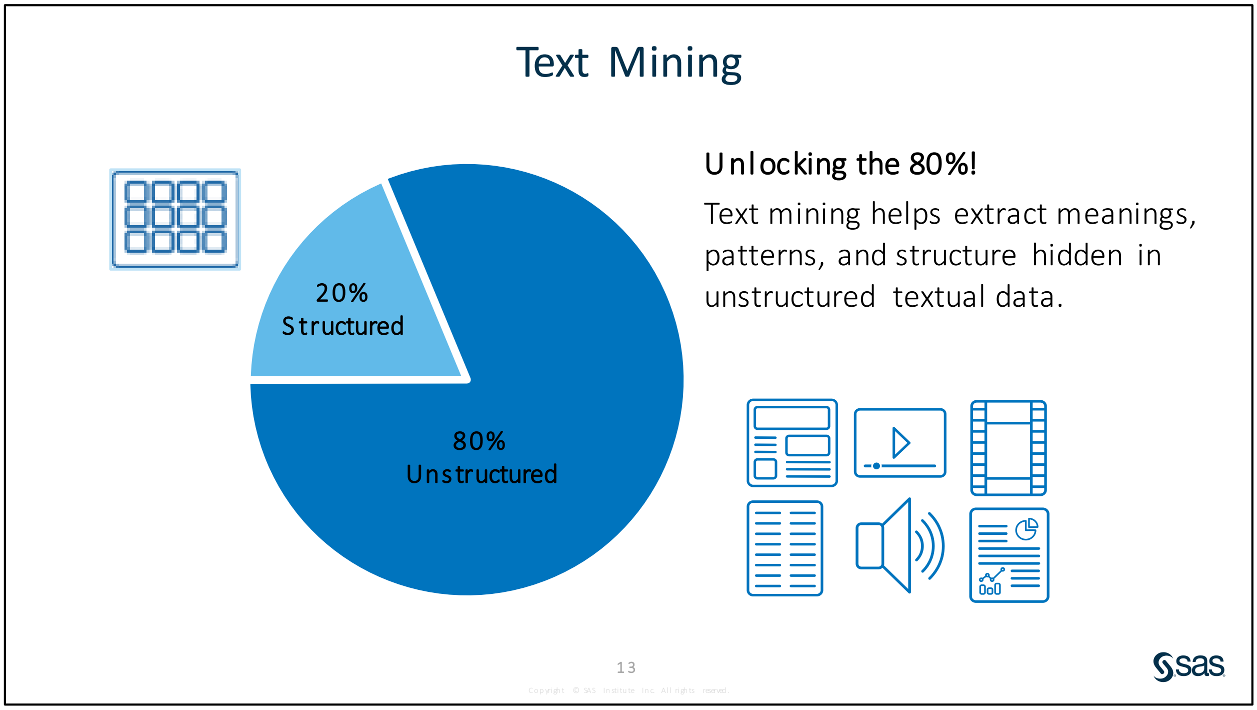
2.2 Feature Extraction



Essential Data Tasks

- Divide the data.
- Address rare events.
- Manage missing values.
- Add unstructured data.
- Extract features.
- Handle extreme or unusual values.
- Select useful inputs.

12



Text Mining

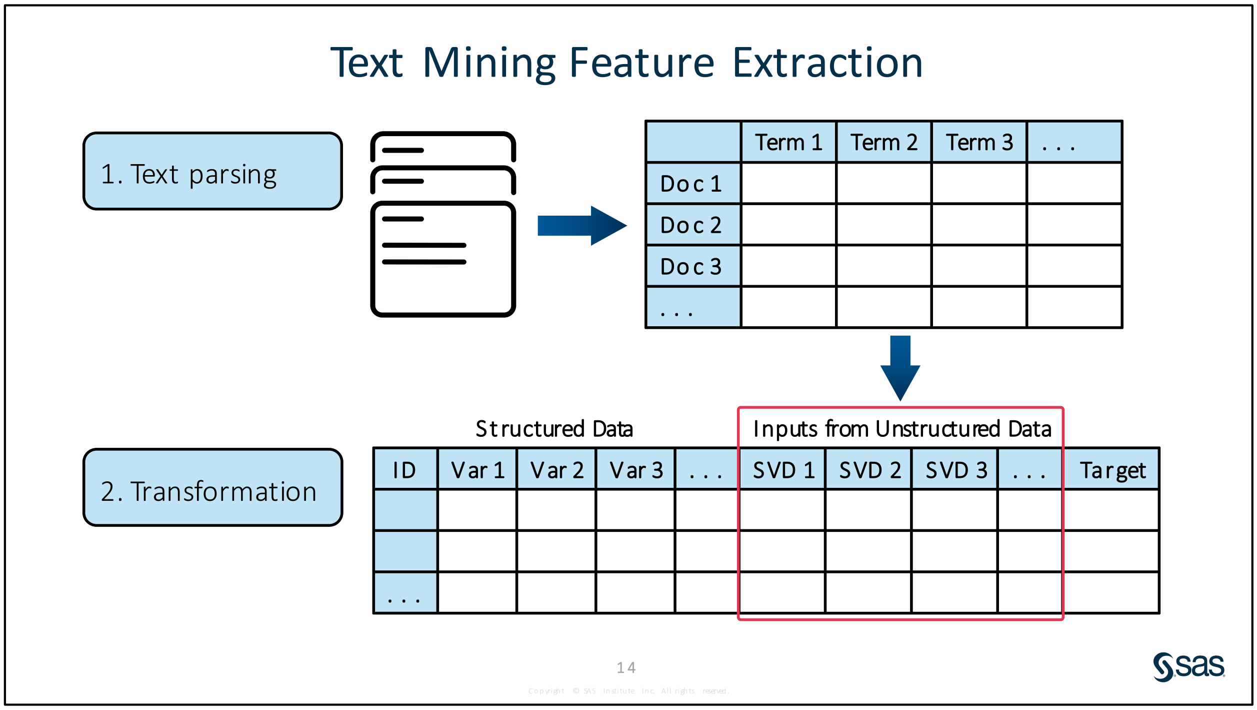
Unlocking the 80%!

Text mining helps extract meanings, patterns, and structure hidden in unstructured textual data.

13

Organizations today are generating and storing tremendous amounts of data. IDC has estimated that up to 80% of that is *unstructured*—that is, information that either does not have a predefined data model or is not organized in a predefined way. Unstructured data includes formats such as audio, images, video, and textual content. Although this type of information is often rich with insights, unlocking the full potential within these complex data sources can be tricky. Much of the big data explosion is due to the rapid growth of unstructured data!

Source: IDC Digital Universe Study, sponsored by EMC, May 2010.



Text Mining Feature Extraction

1. Text parsing

	Term 1	Term 2	Term 3	. . .
Doc 1				
Doc 2				
Doc 3				
. . .				

2. Transformation

Structured Data

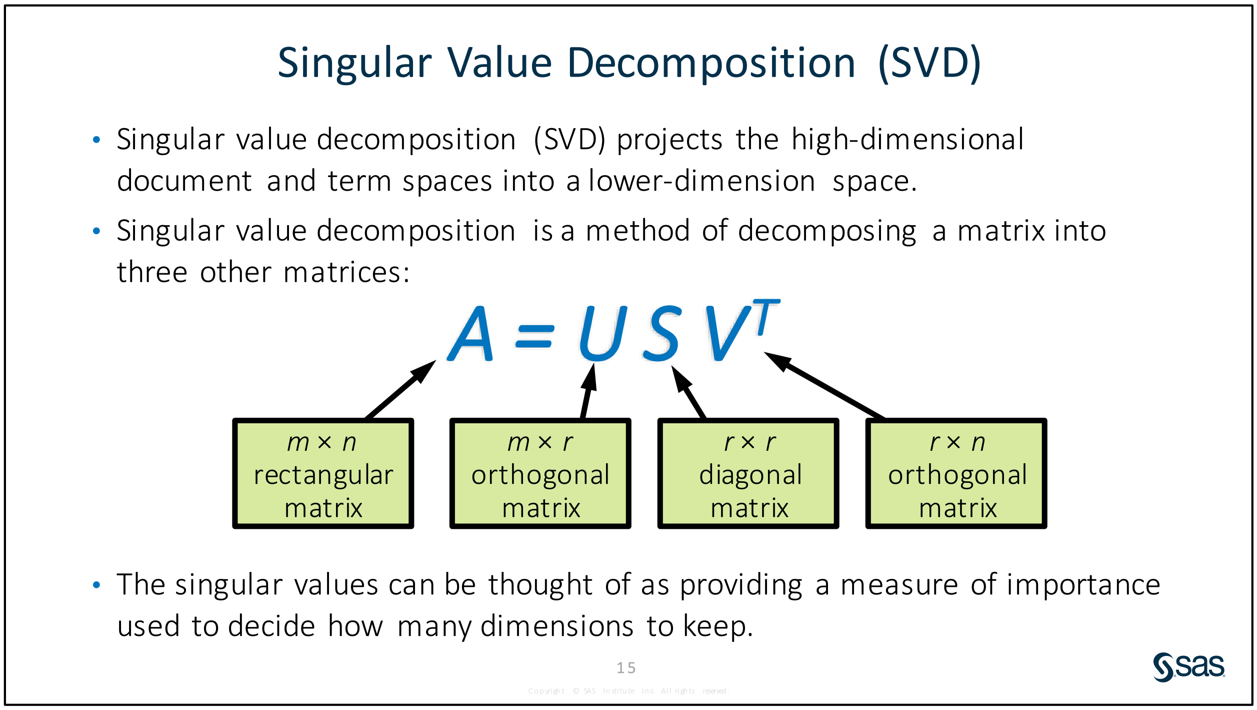
Inputs from Unstructured Data

ID	Var 1	Var 2	Var 3	. . .	SVD 1	SVD 2	SVD 3	. . .	Target
. . .									

14

Unstructured free-form text data are commonly available in business. For example, survey results, call center logs, product reviews, social media feeds, blogs, customer feedback, and other text data contain information useful for predictive modeling outcomes that is not readily available in structured data. It is therefore extremely informative to analyze these combined text data sources and use them along with the structured data.

The Text Mining node in Model Studio enables you to process text data in a document collection. Often, you might be able to improve the predictive ability of your models that use only numerical data if you add selected text mining results (clusters or SVD values) to the numerical data. Data are processed in two phases: text parsing and transformation. Text parsing processes textual data into a term-by-document frequency matrix. Transformations such as singular value decomposition (SVD) alter this matrix into a data set that is suitable for data mining purposes. A document collection with thousands of documents and terms can be represented in a compact and efficient form.



Singular Value Decomposition (SVD)

- Singular value decomposition (SVD) projects the high-dimensional document and term spaces into a lower-dimension space.
- Singular value decomposition is a method of decomposing a matrix into three other matrices:

$$A = U\,S\,V^T$$

$m \times n$ rectangular matrix
$m \times r$ orthogonal matrix
$r \times r$ diagonal matrix
$r \times n$ orthogonal matrix

- The singular values can be thought of as providing a measure of importance used to decide how many dimensions to keep.

15

Consider ***A*** to be a term-document matrix with *m* terms and *n* documents. (Typically, $m > n$. That is, there are more terms than documents.) The SVD theorem states that the term-document matrix (and, in fact, ***any*** rectangular matrix of real or complex values) can always be decomposed into the product of three matrices. Below are the details:

- ***A*** is the real *m* x *n* matrix that you want to decompose.
- ***U*** is an *m* x *r* matrix that contains the left singular vectors and satisfying the orthogonality condition $\boldsymbol{U}^T\boldsymbol{U}=I_{r\times r}$.
- *r* is the rank of the matrix ***A***.
- $I_{r\times r}$ is an *r* x *r* identity matrix.
- ***S*** is an *r* x *r* diagonal matrix consisting of *r* positive "singular values" $s_1 \geq s_2 \geq \ldots \geq s_r > 0$.
- ***V*** is an *r* x *n* matrix that contains the right singular vectors and satisfies the orthogonality condition $\boldsymbol{V}\boldsymbol{V}^T=I_{r\times r}$.
- *T* signifies the transpose of a matrix.

Details: SVD

This brief discussion is based on the very helpful paper "Taming Text with the SVD" (recommended reading and readily available for downloading from the internet) by Dr. Russ Albright of SAS R&D. The most relevant aspects of the SVD theorem are presented for dimension reduction, followed by an example. If you do not understand the abstract explanations, the concrete example at least gives you the "flavor" of what is happening.

Russ Albright's example consists of three documents:

Doc 1: Error: invalid message file format

Doc 2: Error: unable to open message file using message path

Doc 3: Error: unable to format variable

These three documents generate the following 11 x 3 term-document matrix *A*.

		doc 1	doc 2	doc 3
Term 1	error	**1**	**1**	**1**
Term 2	invalid	**1**	**0**	**0**
Term 3	message	**1**	**2**	**0**
Term 4	file	**1**	**1**	**0**
Term 5	format	**1**	**0**	**1**
Term 6	unable	**0**	**1**	**1**
Term 7	to	**0**	**1**	**1**
Term 8	open	**0**	**1**	**0**
Term 9	using	**0**	**1**	**0**
Term 10	path	**0**	**1**	**0**
Term 11	variable	**0**	**0**	**1**

With the right software (for example, PROC IML), it is very easy to compute the SVD decomposition for this example and obtain the separate matrices *U*, *S,* and *V*.

The product U^TA produces the SVD projections of the original document vectors. These are the document SVD input values (COL columns) that you will see in the next demonstration produced by the Text Mining node (except that they are normalized for each document as explained later).

This amounts to forming linear combinations of the original (possibly weighted) term frequencies for each document.

First project the first document vector d_1 into a three- dimensional SVD space by the matrix multiplication:

$$U^T d_1 =$$

$$\begin{vmatrix} 0.43 & 0.11 & 0.55 & 0.33 & 0.21 & 0.31 & 0.31 & 0.22 & 0.22 & 0.22 & 0.09 \\ 0.30 & 0.13 & -0.37 & -0.12 & 0.55 & 0.18 & 0.18 & -0.25 & -0.25 & -0.25 & 0.43 \\ 0.11 & 0.52 & 0.2 & 0.36 & 0.27 & -0.41 & -0.41 & -0.16 & -0.16 & -0.16 & -0.25 \end{vmatrix} * \begin{vmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{vmatrix}$$

U*^{T}* was obtained using the SVD matrix function in PROC IML applied to matrix ***A***.

d_1 is the term-frequency vector for document 1.

The product of the 3 x 11 ***U****^{T}* matrix with the 11 x 1 term-frequency vector d_1 for doc 1 gives the following:

$$U^T d_1 = \hat{d}_1 = \begin{vmatrix} 1.63 \\ 0.49 \\ 1.45 \end{vmatrix}$$

And then write this in transposed form with column labels:

$$\hat{d}_1^{\,T} = \begin{array}{c} \begin{array}{ccc} \text{SVD1} & \text{SVD2} & \text{SVD3} \end{array} \\ \begin{vmatrix} 1.63 & 0.49 & 1.45 \end{vmatrix} \end{array}$$

The SVD dimensions are ordered by the size of their singular values (their *importance*). Therefore, the document vector can simply be truncated to obtain a lower-dimensional projection:

The 2-D representation for doc 1 is $\begin{array}{cc} \text{SVD1} & \text{SVD2} \\ |\,\mathbf{1.63} & \mathbf{0.49}\,| \end{array}$.

As a final step, these coordinate values are normalized so that the sums of squares for each document are 1.0:

Using this document's 2-D representation, $1.63^2 + 0.49^2 = 2.847$ and $\sqrt{2.897} = 1.70$.

Therefore, the final 2-D representation for doc 1 would be $\begin{array}{cc} \text{SVD1} & \text{SVD2} \\ |\,\mathbf{0.96} & \mathbf{0.29}\,| \end{array}$.

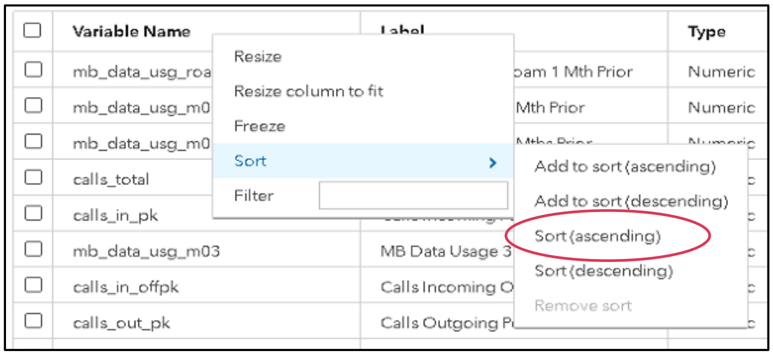
These are the SVD1 and SVD2 values that you would see for this document. A similar calculation is performed for the other two documents.

Adding Text Mining Features

In this demonstration, you create new features using the Text Mining node. The **commsdata** data has five text variables. You use the text variable **verbatims**, which represents free-form, unstructured data from a customer survey.

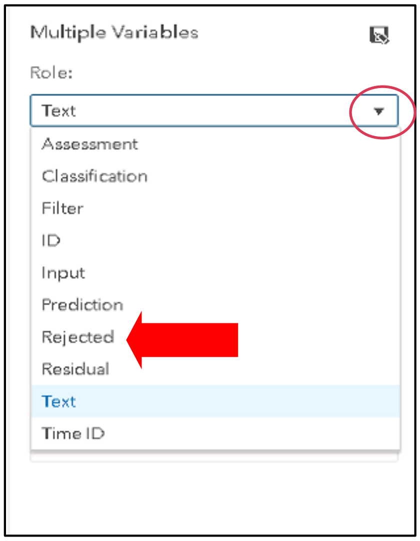
Of the four text variables not used in this demonstration, two are already rejected and two require a metadata change to be rejected. **Call_center** and **issue_level1** already have roles of Rejected, but **issue_level2** and **resolution** need to have their roles changed to Rejected.

1. Click the **Data** tab. Verify that previously selected variables are deselected.
2. Right-click the **Variable Name** column and select **Sort** ⇨ **Sort (ascending)**.



□	Variable Name	Label	Type
□	mb_data_usg_roa	pam 1 Mth Prior	Numeric
□	mb_data_usg_m0	Mth Prior	Numeric
□	mb_data_usg_m0	Mths Prior	Numeric
□	calls_total		
□	calls_in_pk		
□	mb_data_usg_m03	MB Data Usage 3	
□	calls_in_offpk	Calls Incoming O	
□	calls_out_pk	Calls Outgoing P	

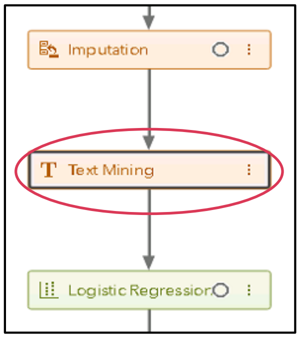
3. Select **issue_level2** and **resolution**. In the pane on the right, change the role from Text to **Rejected**.



This ensures that only the **verbatims** variable is used as an input for the Text Mining node.

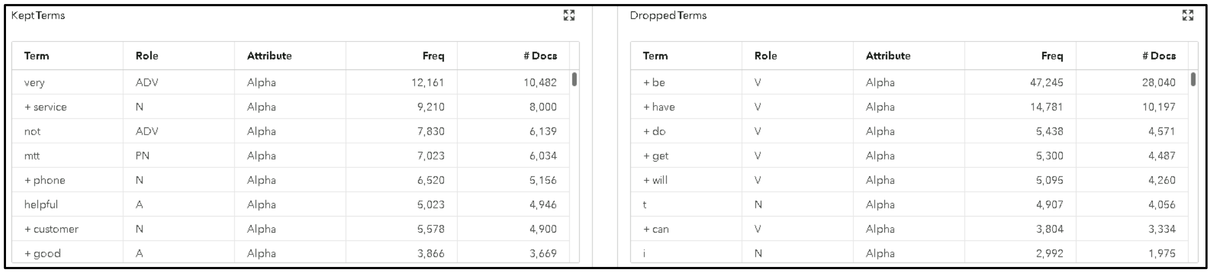
4. Return to the Starter Template pipeline. (Click the **Pipelines** tab and then select **Starter Template**.)

5. From the pane on the left, drag and drop a **Text Mining** node ***between*** the Imputation node and the Logistic Regression node.



6. Right-click the **Text Mining** node and select **Run**.

7. Open the results of the Text Mining node. Many windows are available, including tables of Kept Terms and Dropped Terms. These tables include terms used and ignored, respectively, during the text analysis.



Kept Terms

Term	Role	Attribute	Freq	# Docs
very	ADV	Alpha	12,161	10,482
+ service	N	Alpha	9,210	8,000
not	ADV	Alpha	7,830	6,139
mtt	PN	Alpha	7,023	6,034
+ phone	N	Alpha	6,520	5,156
helpful	A	Alpha	5,023	4,946
+ customer	N	Alpha	5,578	4,900
+ good	A	Alpha	3,866	3,669

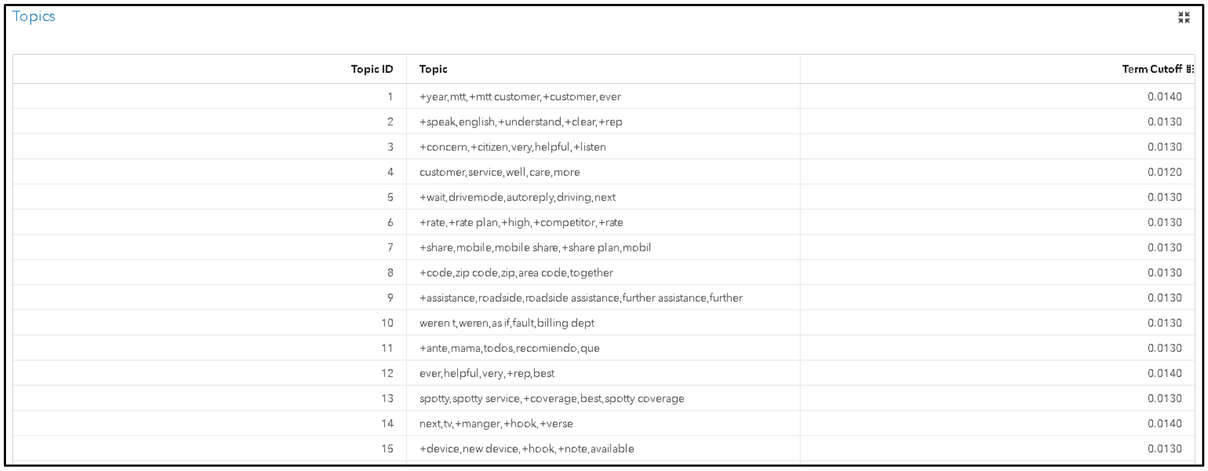
Dropped Terms

Term	Role	Attribute	Freq	# Docs
+ be	V	Alpha	47,245	28,040
+ have	V	Alpha	14,781	10,197
+ do	V	Alpha	5,438	4,571
+ get	V	Alpha	5,300	4,487
+ will	V	Alpha	5,095	4,260
t	N	Alpha	4,907	4,056
+ can	V	Alpha	3,804	3,334
i	N	Alpha	2,992	1,975

Note: The plus sign next to a word indicates stemming (for example, +service represents *service*, *services*, *serviced*, and so on).

8. Expand the **Topics** table. This table shows topics created by the Text Mining node.

Topics are created based on groups of terms that occur together in several documents. Each term-document pair is assigned a score for every topic. Thresholds are then used to determine whether the association is strong enough to consider whether that document or term belongs in the topic. Because of this, terms and documents can belong to multiple topics.

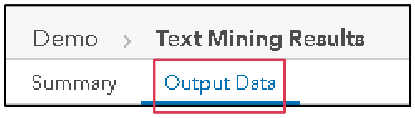


Topics

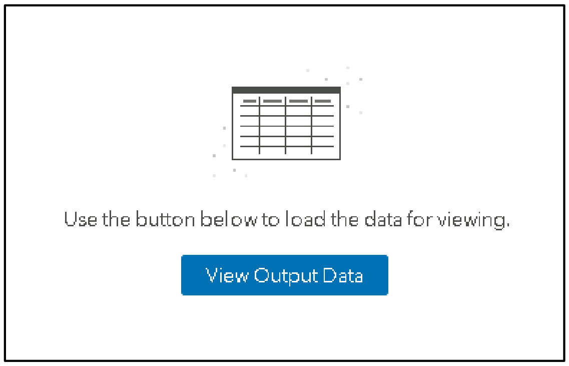
Topic ID	Topic	Term Cutoff
1	+year,mtt,+mtt customer,+customer,ever	0.0140
2	+speak,english,+understand,+clear,+rep	0.0130
3	+concern,+citizen,very,helpful,+listen	0.0130
4	customer,service,well,care,more	0.0120
5	+wait,drivemode,autoreply,driving,next	0.0130
6	+rate,+rate plan,+high,+competitor,+rate	0.0130
7	+share,mobile,mobile share,+share plan,mobil	0.0130
8	+code,zip code,zip,area code,together	0.0130
9	+assistance,roadside,roadside assistance,further assistance,further	0.0130
10	weren t,weren,as if,fault,billing dept	0.0130
11	+ante,mama,todos,recomiendo,que	0.0130
12	ever,helpful,very,+rep,best	0.0140
13	spotty,spotty service,+coverage,best,spotty coverage	0.0130
14	next,tv,+manger,+hook,+verse	0.0140
15	+device,new device,+hook,+note,available	0.0130

Because 15 topics were discovered, 15 new columns of inputs are created. The output columns contain SVD scores that can be used as inputs for the downstream nodes.

9. Click the **Output Data** tab.

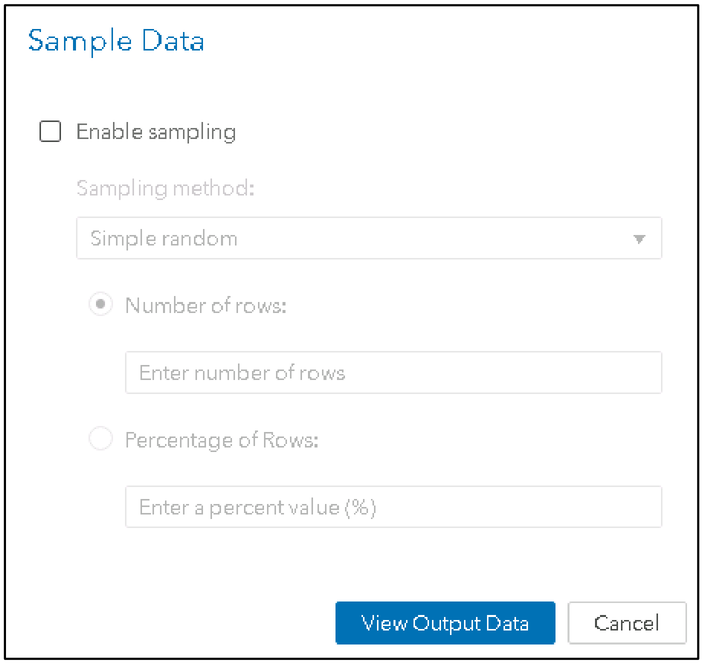


10. Click **View Output Data**.



Use the button below to load the data for viewing.

11. Click **View Output Data** again. In this step, you can choose to create a sample of the data to be viewed.



Sample Data

□ Enable sampling

Sampling method:

Simple random

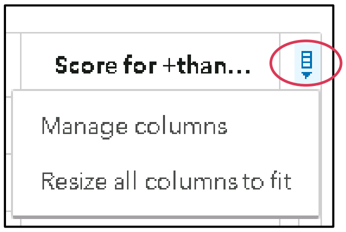
● Number of rows:

Enter number of rows

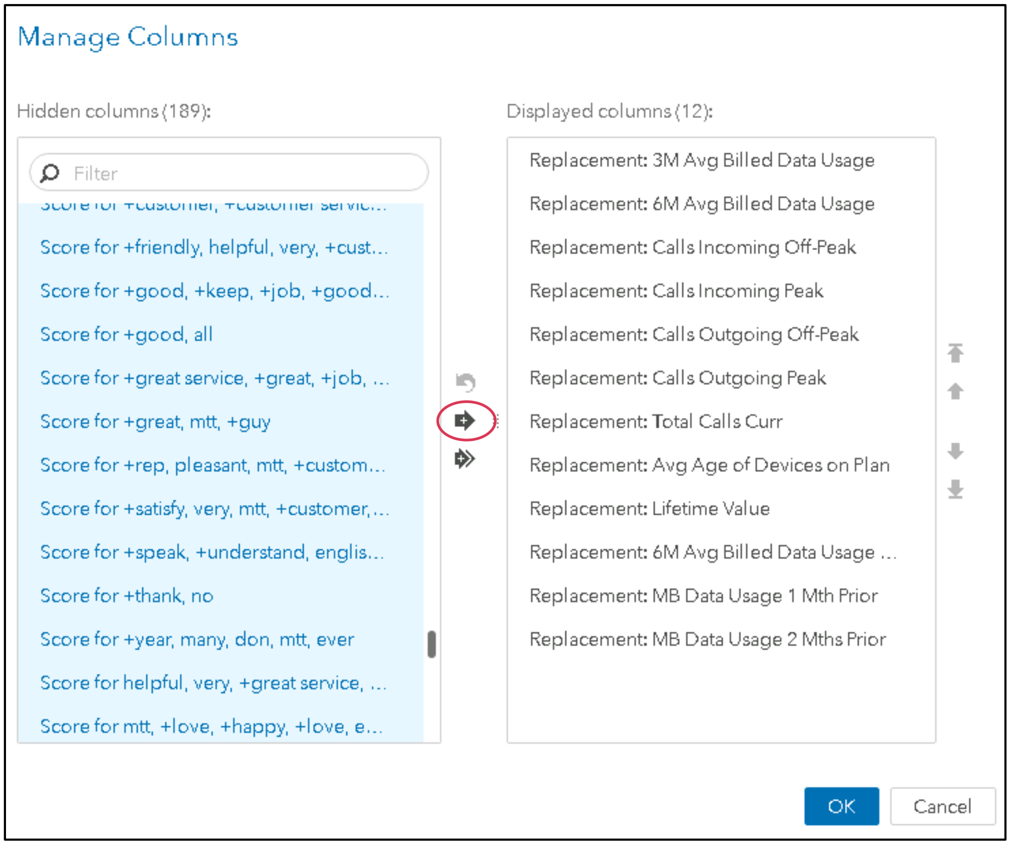
○ Percentage of Rows:

Enter a percent value (%)

12. Click the **Options** button and then select **Manage columns**.



13. In the left pane, select all **Score** variables and click the arrow that has a **plus sign** on it.



Manage Columns

Hidden columns (189):

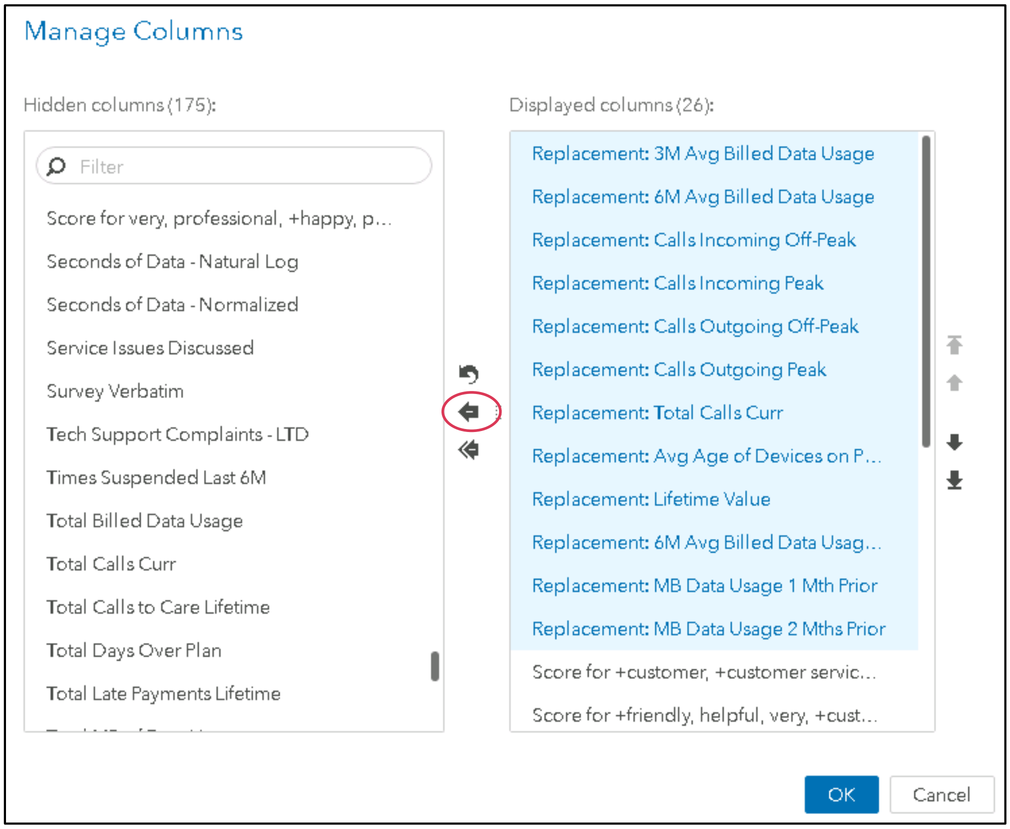
Filter

Score for +customer, +customer servic...
Score for +friendly, helpful, very, +cust...
Score for +good, +keep, +job, +good...
Score for +good, all
Score for +great service, +great, +job, ...
Score for +great, mtt, +guy
Score for +rep, pleasant, mtt, +custom...
Score for +satisfy, very, mtt, +customer, ...
Score for +speak, +understand, englis...
Score for +thank, no
Score for +year, many, don, mtt, ever
Score for helpful, very, +great service, ...
Score for mtt, +love, +happy, +love, e...

Displayed columns (12):

Replacement: 3M Avg Billed Data Usage
Replacement: 6M Avg Billed Data Usage
Replacement: Calls Incoming Off-Peak
Replacement: Calls Incoming Peak
Replacement: Calls Outgoing Off-Peak
Replacement: Calls Outgoing Peak
Replacement: Total Calls Curr
Replacement: Avg Age of Devices on Plan
Replacement: Lifetime Value
Replacement: 6M Avg Billed Data Usage ...
Replacement: MB Data Usage 1 Mth Prior
Replacement: MB Data Usage 2 Mths Prior

14. In the right pane, select all **Replacement** variables and click the arrow that has a minus sign on it.



Manage Columns

Hidden columns (175):

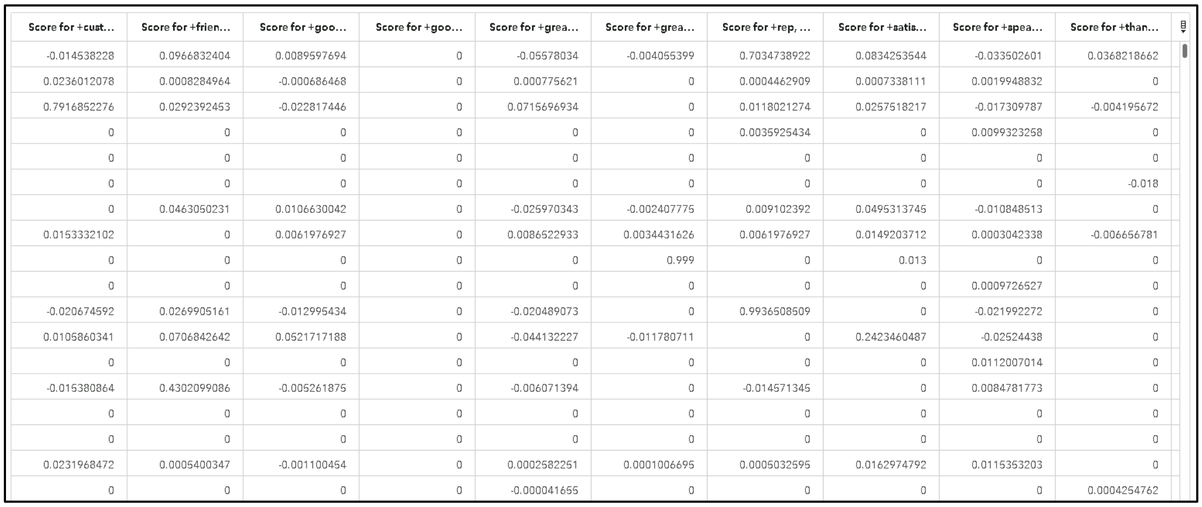
Filter

- Score for very, professional, +happy, p...
- Seconds of Data - Natural Log
- Seconds of Data - Normalized
- Service Issues Discussed
- Survey Verbatim
- Tech Support Complaints - LTD
- Times Suspended Last 6M
- Total Billed Data Usage
- Total Calls Curr
- Total Calls to Care Lifetime
- Total Days Over Plan
- Total Late Payments Lifetime

Displayed columns (26):

- Replacement: 3M Avg Billed Data Usage
- Replacement: 6M Avg Billed Data Usage
- Replacement: Calls Incoming Off-Peak
- Replacement: Calls Incoming Peak
- Replacement: Calls Outgoing Off-Peak
- Replacement: Calls Outgoing Peak
- Replacement: Total Calls Curr
- Replacement: Avg Age of Devices on P...
- Replacement: Lifetime Value
- Replacement: 6M Avg Billed Data Usag...
- Replacement: MB Data Usage 1 Mth Prior
- Replacement: MB Data Usage 2 Mths Prior
- Score for +customer, +customer servic...
- Score for +friendly, helpful, very, +cust...

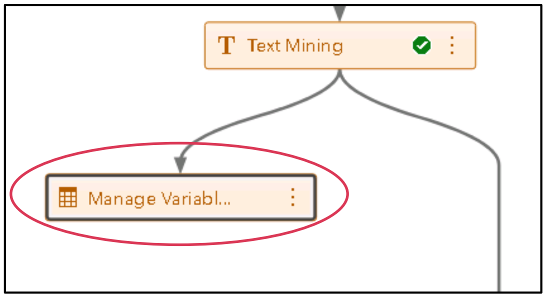
15. Click **OK**. The SVD coefficients (scores) are shown for the 15 topics discovered (columns **COL1** through **COL15**), for each observation in the data set. Those columns are passed along for the following nodes.



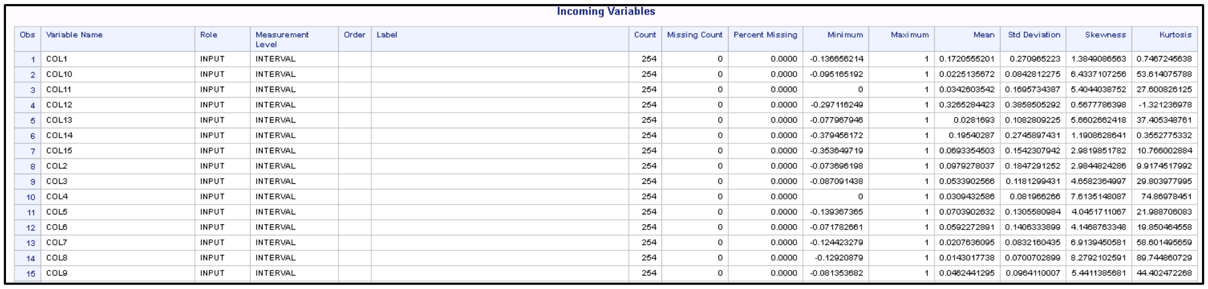
Score for +cust...	Score for +frien...	Score for +goo...	Score for +goo...	Score for +grea...	Score for +grea...	Score for +rep, ...	Score for +satis...	Score for +spea...	Score for +than...
-0.014538228	0.0966832404	0.0089597694	0	-0.05578034	-0.004055399	0.7034738922	0.0834253544	-0.033502601	0.0368218662
0.0236012078	0.0008284964	-0.000686468	0	0.000775621	0	0.0004462909	0.0007338111	0.0019948832	0
0.7916852276	0.0292392453	-0.022817446	0	0.0715696934	0	0.0118021274	0.0257518217	-0.017309787	-0.004195672
0	0	0	0	0	0	0.0035925434	0	0.0099323258	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	-0.018
0	0.0463050231	0.0106630042	0	-0.025970343	-0.002407775	0.009102392	0.0495313745	-0.010848513	0
0.0153332102	0	0.0061976927	0	0.0086522933	0.0034431626	0.0061976927	0.0149203712	0.0003042338	-0.006656781
0	0	0	0	0	0.999	0	0.013	0	0
0	0	0	0	0	0	0	0	0.0009726527	0
-0.020674592	0.0269905161	-0.012995434	0	-0.020489073	0	0.9936508509	0	-0.021992272	0
0.0105860341	0.0706842642	0.0521717188	0	-0.044132227	-0.011780711	0	0.2423460487	-0.02524438	0
0	0	0	0	0	0	0	0	0.0112007014	0
-0.015380864	0.4302099086	-0.005261875	0	-0.006071394	0	-0.014571345	0	0.0084781773	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0.0231968472	0.0005400347	-0.001100454	0	0.0002582251	0.0001006695	0.0005032595	0.0162974792	0.0115353203	0
0	0	0	0	-0.000041655	0	0	0	0	0.0004254762

16. Close the Results window.

17. Alternatively, use the Manage Variables node to see that 15 new interval input columns were added to the data. Right-click the **Text Mining** node and select **Add below** ⇨ **Data Mining Preprocessing** ⇨ **Manage Variables**.



18. Run the **Manage Variables** node and view the results when it is complete. Expand the Output window. At the top of the Incoming Variables table are the 15 new columns representing the dimensions of the SVD calculations based on the 15 topics discovered by the Text Mining node.



Incoming Variables

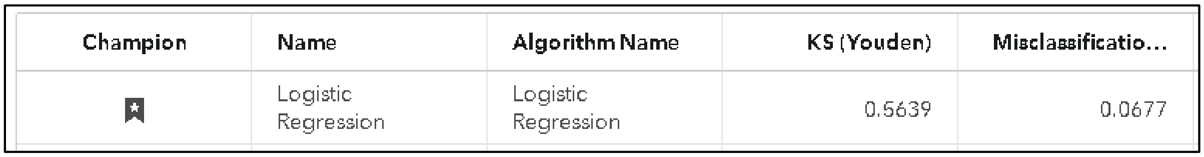
Obs	Variable Name	Role	Measurement Level	Order	Label	Count	Missing Count	Percent Missing	Minimum	Maximum	Mean	Std Deviation	Skewness	Kurtosis
1	COL1	INPUT	INTERVAL			254	0	0.0000	-0.136656214	1	0.1720555201	0.270965223	1.3849086563	0.7467245638
2	COL10	INPUT	INTERVAL			254	0	0.0000	-0.095165192	1	0.0225135672	0.0842812275	6.4337107256	53.614075788
3	COL11	INPUT	INTERVAL			254	0	0.0000	0	1	0.0342603542	0.1695734387	5.4044038752	27.600826125
4	COL12	INPUT	INTERVAL			254	0	0.0000	-0.297116249	1	0.3265284423	0.3858505292	0.5677786398	-1.321236978
5	COL13	INPUT	INTERVAL			254	0	0.0000	-0.077967946	1	0.0281693	0.1082809225	5.6602662418	37.405348761
6	COL14	INPUT	INTERVAL			254	0	0.0000	-0.379456172	1	0.19540287	0.2745897431	1.1908628641	0.3552775332
7	COL15	INPUT	INTERVAL			254	0	0.0000	-0.353649719	1	0.0693354503	0.1542307942	2.9819851782	10.766002884
8	COL2	INPUT	INTERVAL			254	0	0.0000	-0.073696198	1	0.0979278037	0.1847291252	2.9844824286	9.9174517992
9	COL3	INPUT	INTERVAL			254	0	0.0000	-0.087091438	1	0.0533902566	0.1181299431	4.6582364997	29.803977995
10	COL4	INPUT	INTERVAL			254	0	0.0000	0	1	0.0309432586	0.081966266	7.6135148087	74.86978451
11	COL5	INPUT	INTERVAL			254	0	0.0000	-0.139367365	1	0.0703902632	0.1305580984	4.0451711067	21.988706083
12	COL6	INPUT	INTERVAL			254	0	0.0000	-0.071782661	1	0.0592272891	0.1406333899	4.1468763348	19.850464558
13	COL7	INPUT	INTERVAL			254	0	0.0000	-0.124423279	1	0.0207636095	0.0832160435	6.9139450581	58.601495659
14	COL8	INPUT	INTERVAL			254	0	0.0000	-0.12920879	1	0.0143017738	0.0700702899	8.2792102591	89.744860729
15	COL9	INPUT	INTERVAL			254	0	0.0000	-0.081353682	1	0.0462441295	0.0964110007	5.4411385681	44.402472268

These 15 columns (**COL1** through **COL15**) serve as new, interval inputs for subsequent models.

19. Restore the view of the Output window and close the results.

20. To run the entire pipeline, click the **Run pipeline** button.

21. Open the results of the Model Comparison node.



Champion	Name	Algorithm Name	KS (Youden)	Misclassificatio...
🔖	Logistic Regression	Logistic Regression	0.5639	0.0677

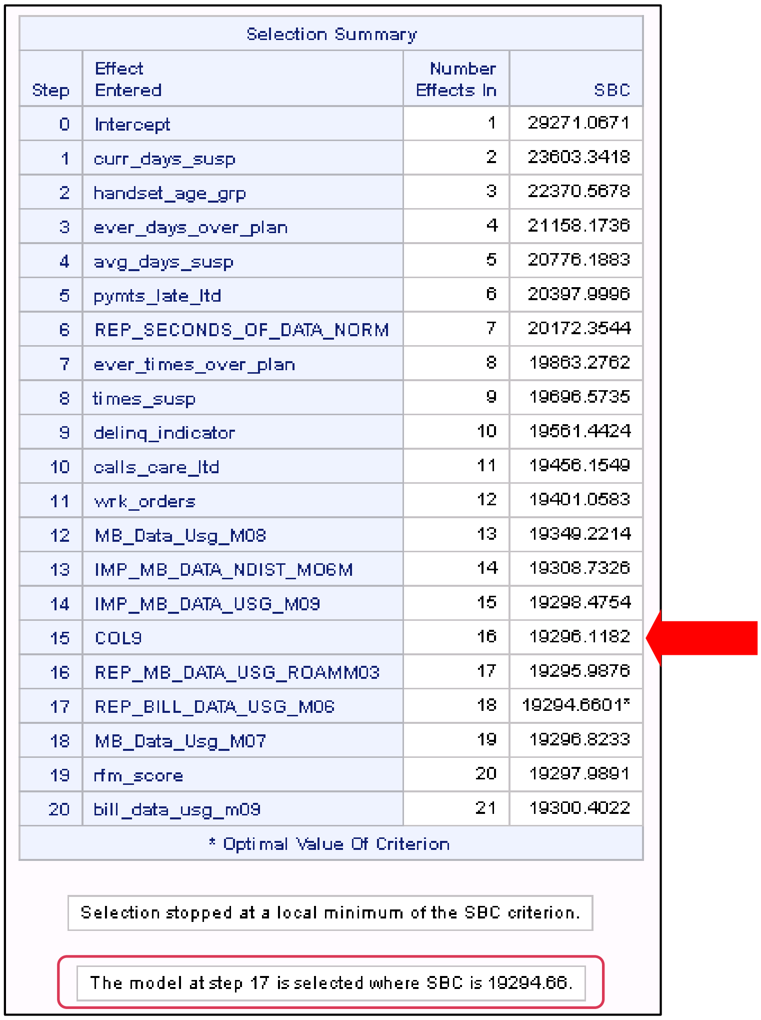
The model does not necessarily improve. Explore the results of the final regression model and see whether it contains one of the text variables.

22. Close the results of the Model Comparison node.

23. Open the results of the Logistic Regression model. Recall that this regression model is based on stepwise selection.

24. Scroll down in the results until you see the Output window. Expand the Output window.

25. Scroll down in the Output window until you see the Selection Summary table. This table shows that one of the columns created by the Text Mining node (**COL9**) did enter the model during the stepwise selection process and remained in the model after optimization of complexity on the hold out sample. This variable entered in step 15 and the final model is from step 17, based on minimum SBC.



Selection Summary

Step	Effect Entered	Number Effects In	SBC
0	Intercept	1	29271.0671
1	curr_days_susp	2	23603.3418
2	handset_age_grp	3	22370.5678
3	ever_days_over_plan	4	21158.1736
4	avg_days_susp	5	20776.1883
5	pymts_late_ltd	6	20397.9996
6	REP_SECONDS_OF_DATA_NORM	7	20172.3544
7	ever_times_over_plan	8	19863.2762
8	times_susp	9	19696.5735
9	delinq_indicator	10	19561.4424
10	calls_care_ltd	11	19456.1549
11	wrk_orders	12	19401.0583
12	MB_Data_Usg_M08	13	19349.2214
13	IMP_MB_DATA_NDIST_MO6M	14	19308.7326
14	IMP_MB_DATA_USG_M09	15	19298.4754
15	COL9	16	19296.1182
16	REP_MB_DATA_USG_ROAMM03	17	19295.9876
17	REP_BILL_DATA_USG_M06	18	19294.6601*
18	MB_Data_Usg_M07	19	19296.8233
19	rfm_score	20	19297.9891
20	bill_data_usg_m09	21	19300.4022

* Optimal Value Of Criterion

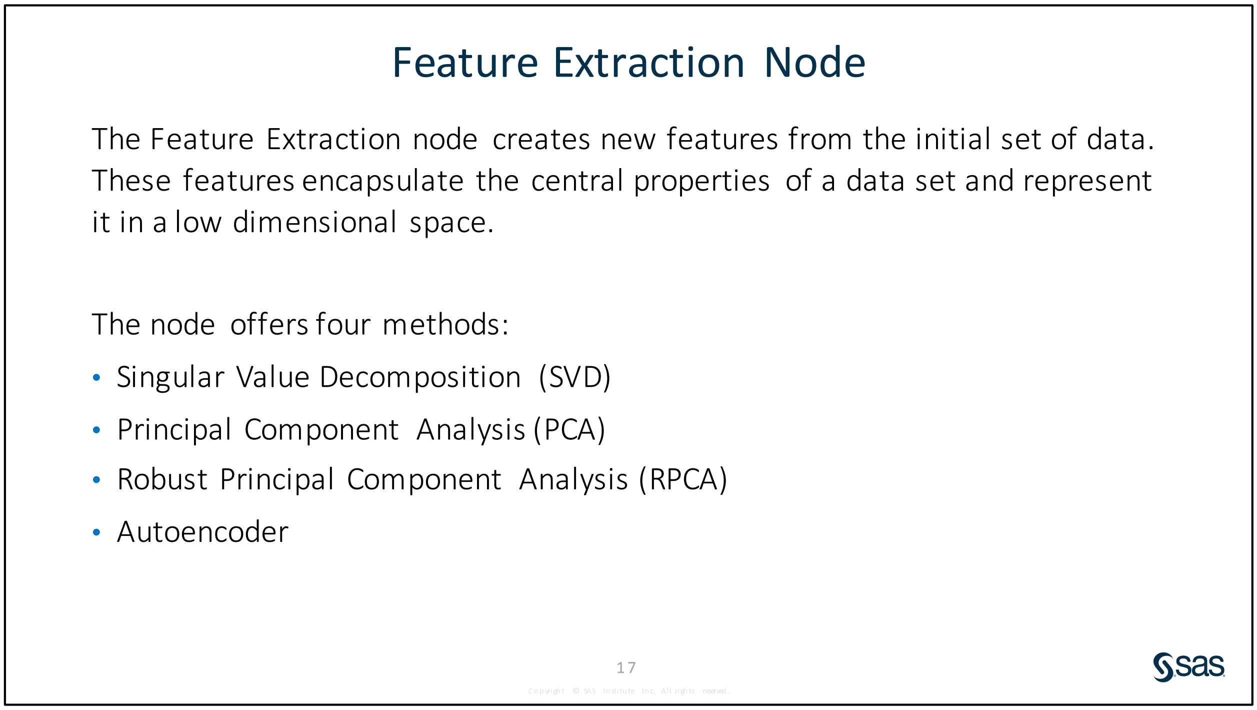
Selection stopped at a local minimum of the SBC criterion.

The model at step 17 is selected where SBC is 19294.66.

26. Restore the view of the Output window and close the Results window.

End of Demonstration

Other Options: The Feature Extraction Node (Self-Study)



Feature Extraction Node

The Feature Extraction node creates new features from the initial set of data. These features encapsulate the central properties of a data set and represent it in a low dimensional space.

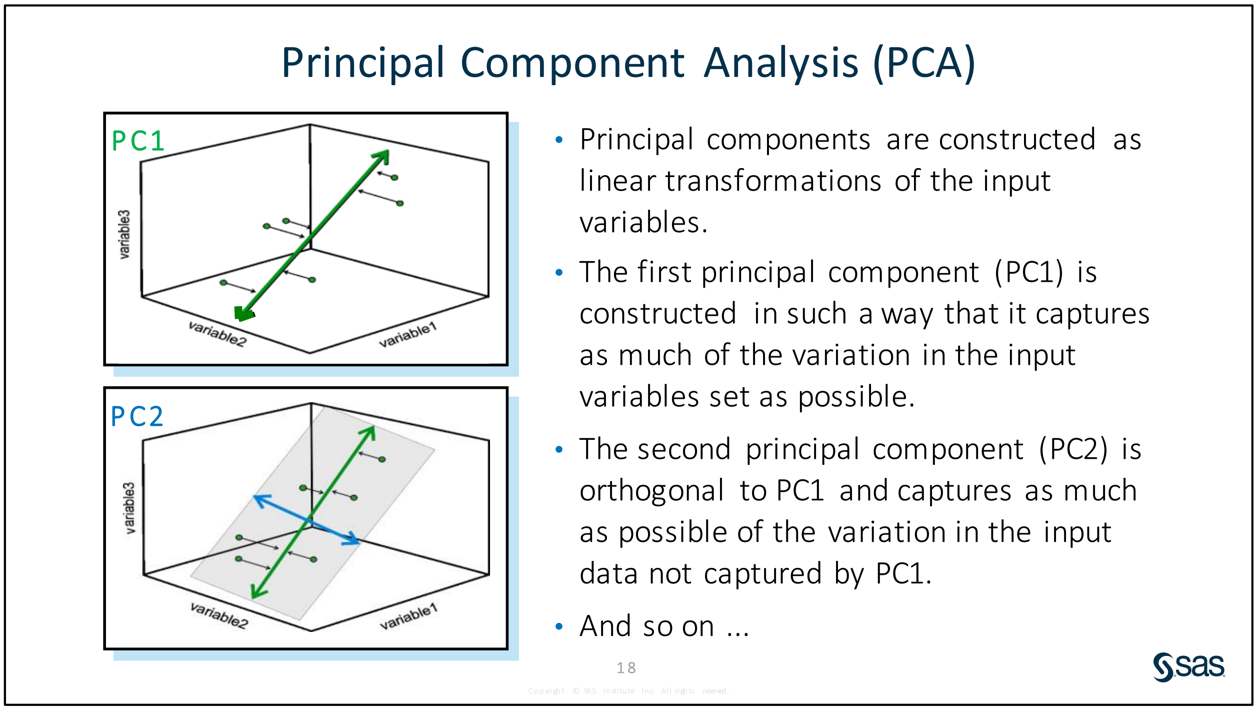
The node offers four methods:

- Singular Value Decomposition (SVD)
- Principal Component Analysis (PCA)
- Robust Principal Component Analysis (RPCA)
- Autoencoder

17

Apart from the text data, the initial (structured) data set of raw features might be too large and unwieldy to be effectively managed, requiring an unreasonable amount of computing resources. Alternatively, the data set might be too robust, causing a classification algorithm to overfit, and providing poor extrapolation in the event of new observations. In either case, the Feature Extraction node can be used to provide a more manageable, representative subset of input variables.

Feature extraction is the process of transforming the existing features into a lower-dimensional space, typically generating new features that are composites of the existing features. SVD is such a technique and has already been discussed. There are many other techniques that reduce dimensionality through such a transformation process, including those discussed on the following slides and pages.



Principal Component Analysis (PCA)

- Principal components are constructed as linear transformations of the input variables.
- The first principal component (PC1) is constructed in such a way that it captures as much of the variation in the input variables set as possible.
- The second principal component (PC2) is orthogonal to PC1 and captures as much as possible of the variation in the input data not captured by PC1.
- And so on ...

18

You can deploy a variety of techniques to correct for (or perhaps more accurately, take advantage of) the distribution flattening. One of the most common statistical approaches is that of *principal component analysis* (PCA). PCA attempts to find a series of orthogonal vectors that better describe the directions of variation in the data than the original inputs do. (A geometric interpretation of orthogonal is that the vectors are perpendicular; a statistical interpretation is that the vectors are uncorrelated.) The goal is to be able to characterize most of the variation in the data with as few vectors as possible.

PCA starts by searching the data's standardized joint distribution for the direction of maximum variation. When found, this direction is labeled the *first principal component*, or *first eigenvector*.

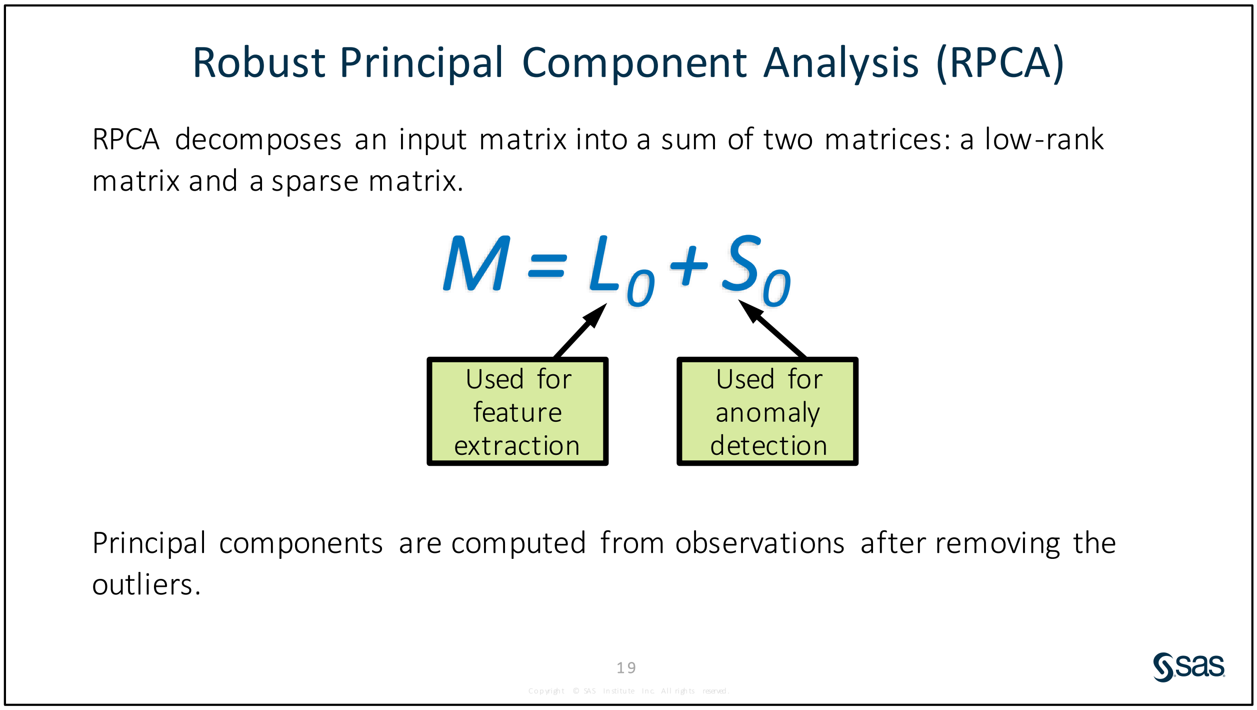
The effect of the first principal component can be removed by projecting the data to a lower dimensional subspace perpendicular to the first principal component.

The difference between the dimension of the original distribution (that is, the number of inputs) and the effective dimension of the projected points is called the *first eigenvalue*.

The data, projected to remove variation in the direction of the first principal component, is again searched for the direction of maximum variation. When identified, this direction is labeled the *second principal component*. The corresponding second eigenvalue can be calculated by again projecting (the data already projected in the first step) along the direction of the second principal component and determining the difference in dimension between the once and twice projected data.

The process of identifying directions of variability, projecting, and calculating eigenvalues continues until the sum of the eigenvalues calculated at each step is close to the dimension of the original input space. This is a common stopping rule: ***How many unique components exist in the data?*** There are as many components as it takes to ensure that the sum of the eigenvalues is greater than 80% or 90% of the input count.

In the presence of redundant inputs, most of the data variability can be described by a few independent principal component vectors.



Robust Principal Component Analysis (RPCA)

RPCA decomposes an input matrix into a sum of two matrices: a low-rank matrix and a sparse matrix.

$$M = L_0 + S_0$$

Used for feature extraction

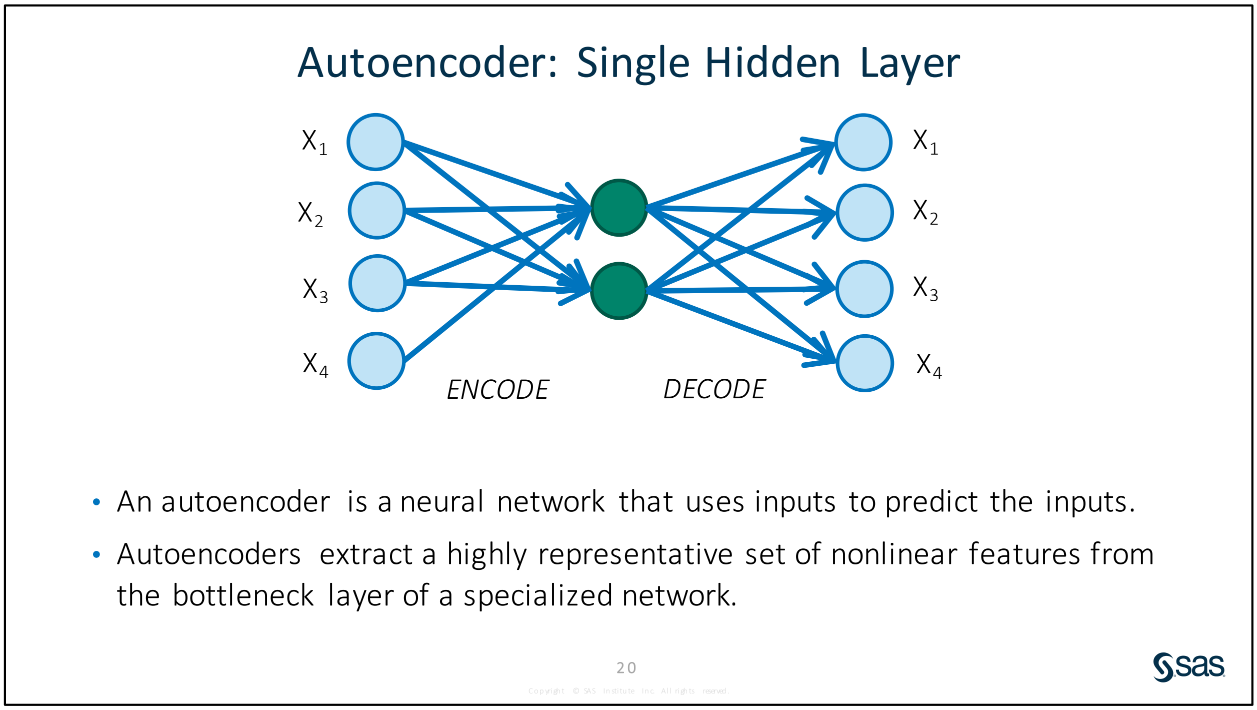
Used for anomaly detection

Principal components are computed from observations after removing the outliers.

19

Robust principal component analysis (RPCA) is a matrix decomposition algorithm that decomposes an input matrix $\boldsymbol{M}$ into a low-rank matrix L_0 and a sparse matrix $\boldsymbol{S}_0$, where $\boldsymbol{M} = L_0 + S_0$. This decomposition is obtained by solving a convex programming problem called principal component pursuit (PCP). The aim in the robust principal component analysis (RPCA) is to recover a low-rank matrix $\boldsymbol{L}_0$ from highly corrupted measurements $\boldsymbol{M}$. Unlike the small noise term N_0 in classical PCA, the entries in $\boldsymbol{S}_0$ can have arbitrarily large magnitude, and their support is assumed to be sparse but unknown. You can use the low-rank matrix $\boldsymbol{L}_0$ to do feature extraction and use the sparse matrix S_0 to detect anomalies. Robustness in RPCA comes from the property that the principal components are computed from observations after removing the outliers—that is, from the low-rank matrix.

There are many applications of RPCA focused on the low-rank matrix, including image processing, latent semantic indexing, ranking, and matrix completion (Candès et al. 2011). Similarly, there are many applications of RPCA focused on the sparse matrix. One example is the extraction of moving objects from the background in surveillance videos.



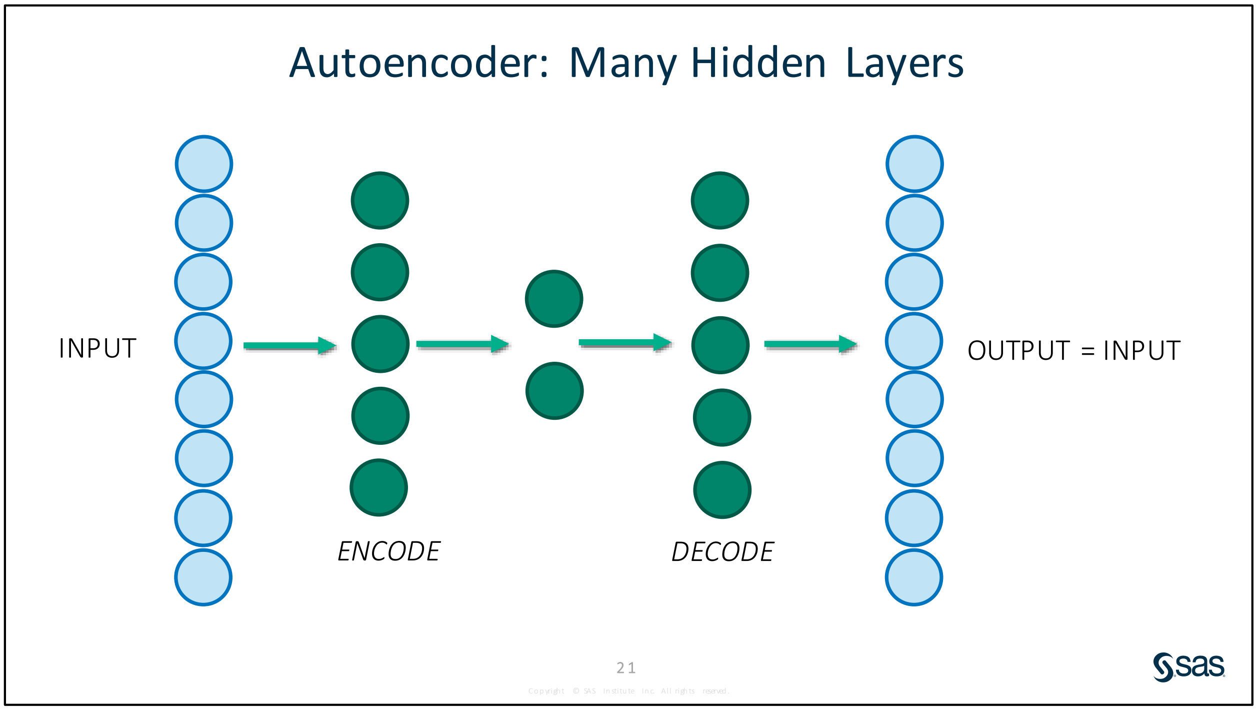
Autoencoder: Single Hidden Layer

- An autoencoder is a neural network that uses inputs to predict the inputs.
- Autoencoders extract a highly representative set of nonlinear features from the bottleneck layer of a specialized network.

20

An autoencoder is a neural network that is used for efficient codings and widely used for feature extraction and nonlinear principal component analysis. Architecturally, an autoencoder is like a multilayer perceptron neural network because it has an input layer, hidden layers (encoding layers), and an output layer (decoding layer). However, it differs in that the output layer is duplicated from the input layer. Therefore, autoencoders are unsupervised learning models. The network is trained to reconstruct its inputs, which forces the hidden layer to try to learn good representations of the inputs.

Autoencoders are like PCA but are much more flexible than PCA. Autoencoders can represent both linear and nonlinear transformation in encoding, but PCA can perform only linear transformation. Autoencoders can be layered to form deep learning network due to its network representation.

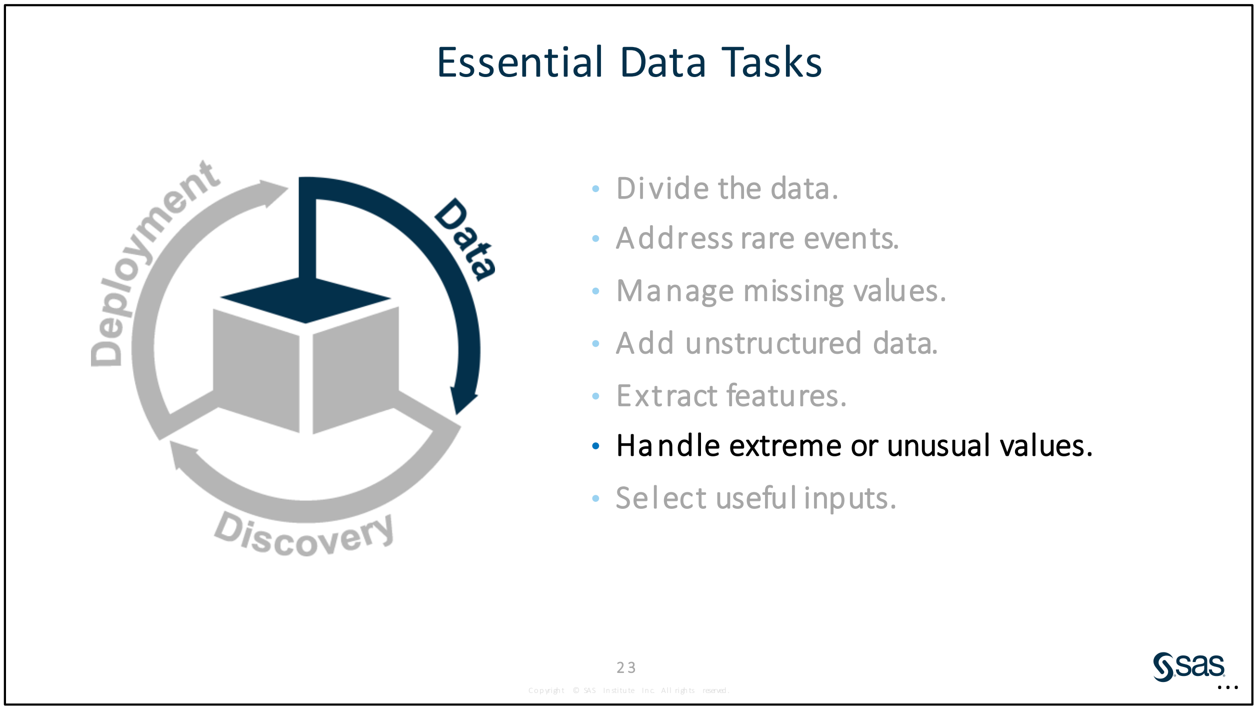


Autoencoder: Many Hidden Layers

For greater network flexibility, we often use more hidden layers with many nodes, like the one above. Each layer of the deep network is usually trained separately by using the output of the previous layer, or by using the training inputs in the case of the first layer. The weights of the individually trained layers are then used to initialize the entire deep network, and all layers are trained again simultaneously on the original training examples. When many inputs are used in conjunction with a much smaller number of hidden units, the features that are extracted as outputs of the hidden units are a nonlinear projection of the training examples onto a lower-dimensional space. Such features can be highly predictive of a training example's class label.

Note: An obvious drawback to feature extraction is that the actual inputs to the model are no longer meaningful with respect to the business problem. However, you can simply consider this another transformation of the original inputs to be provided to the model, something that must be accounted for as part of the scoring process when the model is deployed.

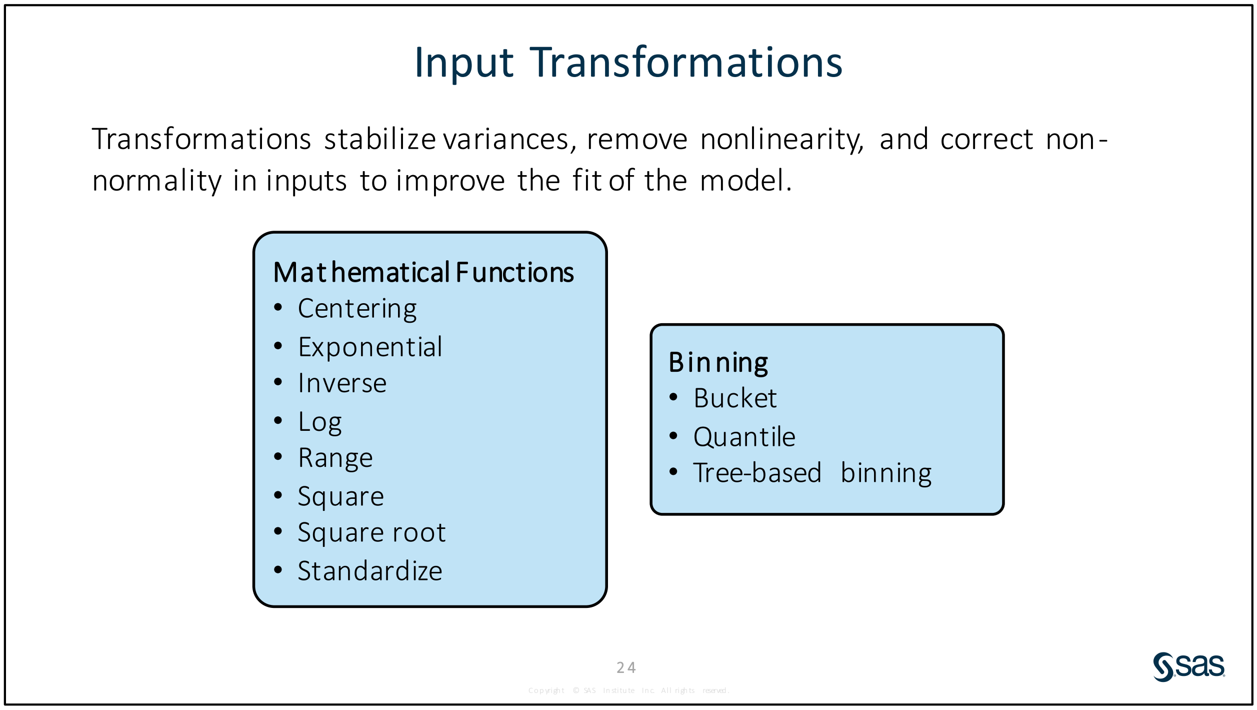
2.3 Input Transformations



Essential Data Tasks

- Divide the data.
- Address rare events.
- Manage missing values.
- Add unstructured data.
- Extract features.
- Handle extreme or unusual values.
- Select useful inputs.

23



Input Transformations

Transformations stabilize variances, remove nonlinearity, and correct non-normality in inputs to improve the fit of the model.

Mathematical Functions
- Centering
- Exponential
- Inverse
- Log
- Range
- Square
- Square root
- Standardize

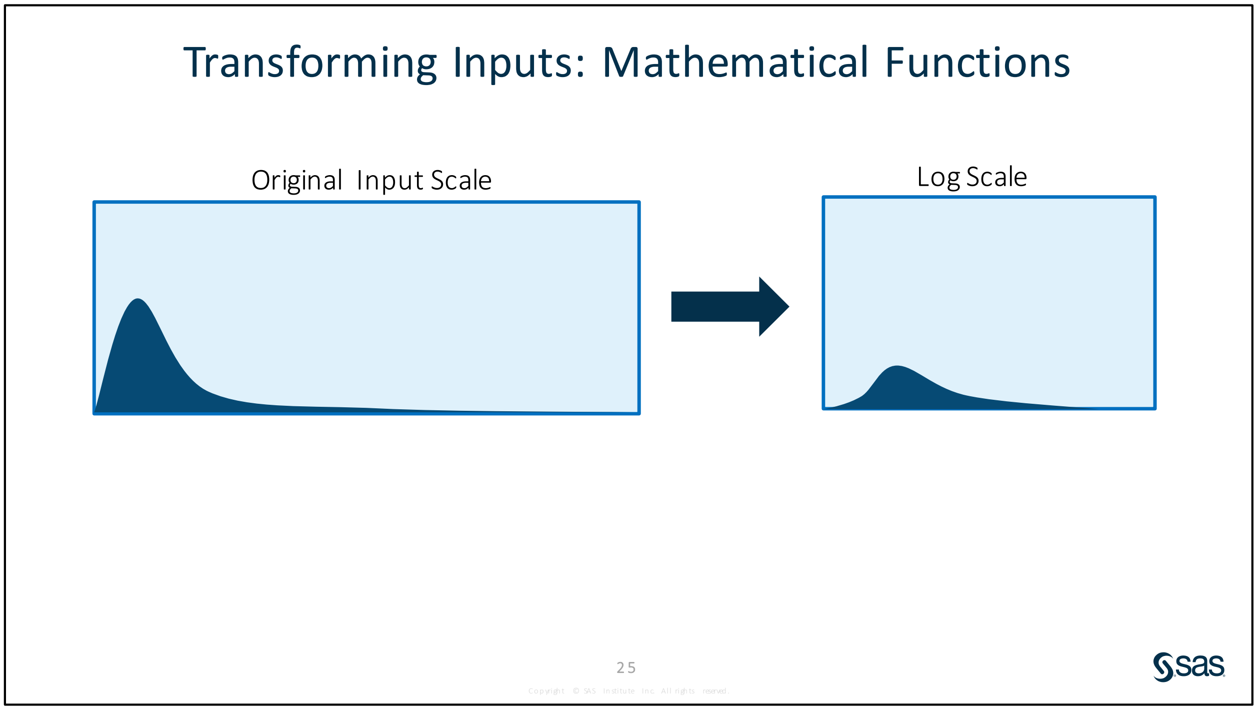
Binning
- Bucket
- Quantile
- Tree-based binning

24

Transformations can be done to change the shape of the distribution of a variable by stretching or compressing it, to reduce the effect of outliers or heavy tails, or to standardize inputs to be on the same range and scale. Another major reason that transformations of inputs are done is to reduce the bias in model predictions.

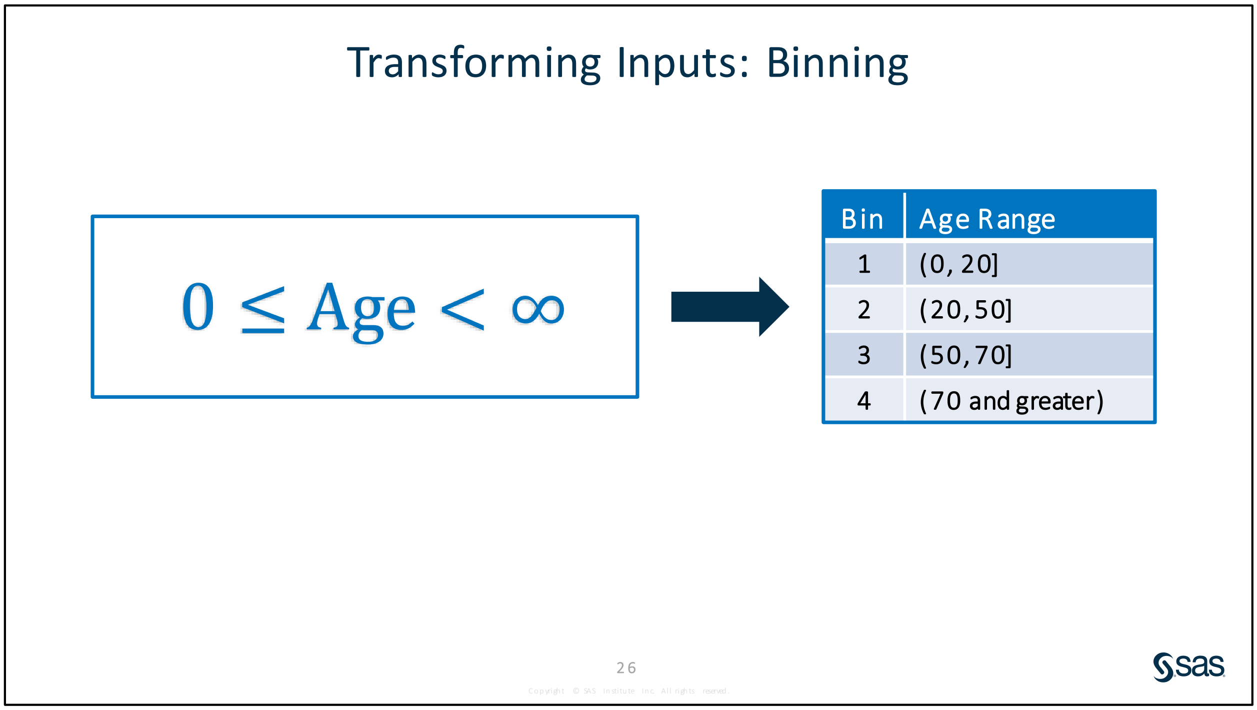
Transformation of input variables is a common data preprocessing task. In machine learning, two types of variable transformations are commonly used:

- mathematical transformations such as square, square root, log or inverse
- binning such as bucket, quantile, or tree-based binning



Transforming Inputs: Mathematical Functions

The simple illustration in the slide above shows a variable distribution, which is positively skewed. The log transformation reduces skewness in it. A distribution that is symmetric or nearly so is often easier to handle and interpret than a skewed distribution. Extreme input distributions are often problematic in predictive modeling. A simpler and, arguably, more effective approach transforms or regularizes offending inputs to eliminate extreme values. Then, a predictive model can be accurately fit using the transformed input in place of the original input. This not only mitigates the influence of extreme cases, but also creates the desired asymptotic association between input and target on the original input scale.



Transforming Inputs: Binning

$0 \leq \text{Age} < \infty$ →

Bin	Age Range
1	(0, 20]
2	(20,50]
3	(50,70]
4	(70 and greater)

26

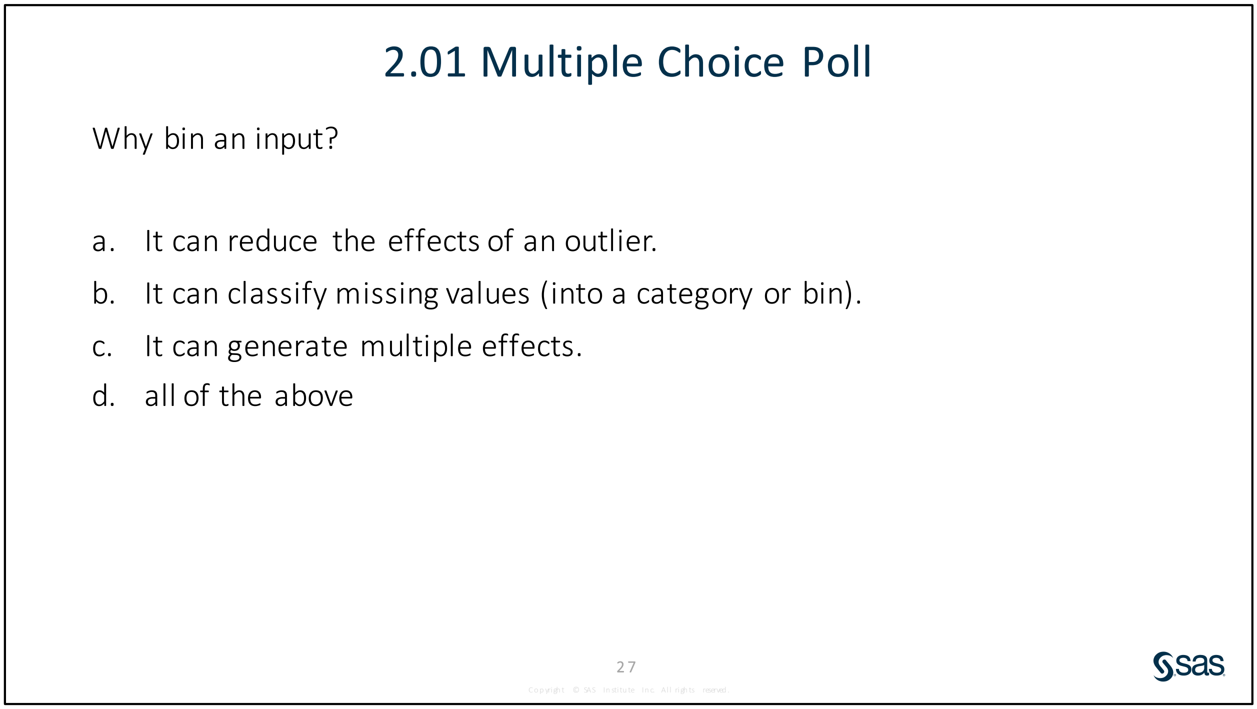
The simple illustration in the slide above shows a variable **Age**, which ranges from 0 to infinity. (In practice, **Age** would not identically equal zero nor approach infinity, but the example is to illustrate how a continuous variable with a large range could be converted to bins.) The **Age** variable is converted into a new variable that takes on only four values, represented by the bins 1 through 4. When the original **Age** variable falls into a certain age range, the binned version of **Age** simply takes the value of the bin that it falls into.

There are many ways that binning can be done. In one case, the bins themselves are of equal width, but the frequency count within each bin can then be varied. Another approach is to make the width of the bins different but the frequency count of observations in each bin consistent.

Binning can be done for several reasons. Binning can be used to classify missing values of a variable, reduce the effect that outliers might have on a model, or illustrate nonlinear relationships between variables. A binned version of a variable also has less variance than the original numeric variable.

Note: **The 'Best' transformation in Model Studio.** "Best" is not really a transformation, but a method or process to select the best transformation for an interval input. In the Transformations node (below), this method is accessed by selecting "Best" via the Default interval inputs method property. When specified, the Best method is applied to all interval inputs coming into the node, unless over-ridden by specific variable transformations identified in metadata via the Data tab or Manage Variables node.

For more information see Best transformation – What is it? at https://communities.sas.com/t5/SAS-Communities-Library/Best-transformation-a-new-feature-in-SAS-Model-Studio-8-3/ta-p/489604.



2.01 Multiple Choice Poll

Why bin an input?

a. It can reduce the effects of an outlier.

b. It can classify missing values (into a category or bin).

c. It can generate multiple effects.

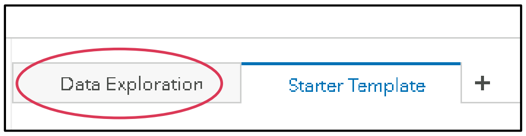
d. all of the above

27

Transforming Inputs

In this demonstration, you use the Transformations node to apply a numerical transformation to input variables.

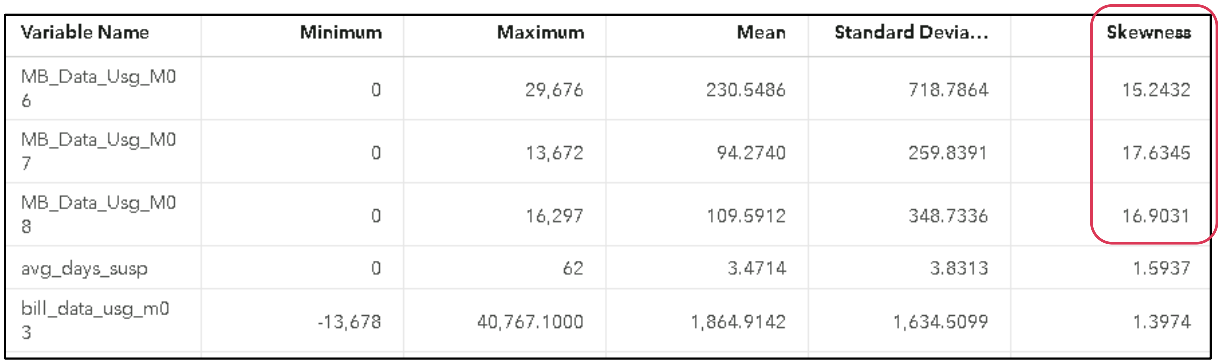
1. Open the **Data Exploration** pipeline by clicking on its tab.



2. Run the **Data Exploration** node. (The pipeline requires a re-run because metadata rules have been applied on the Data tab.)

3. Right-click the **Data Exploration** node and select **Results**.

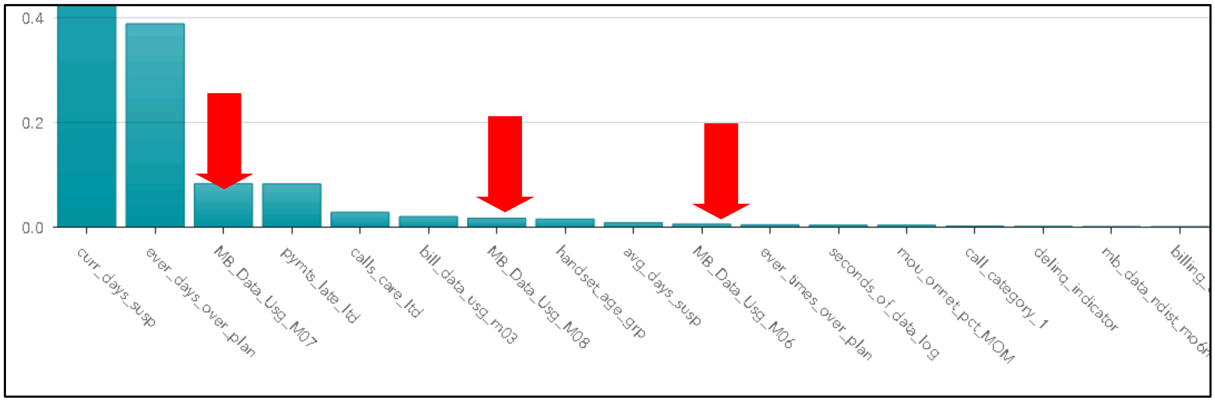
4. Expand the **Interval Variable Moments** table. Note that three of the **MB_Data_Usg_M** variables have a high degree of skewness. Why are only three listed? There is a total of six in the data set.



Variable Name	Minimum	Maximum	Mean	Standard Devia...	Skewness
MB_Data_Usg_M06	0	29,676	230.5486	718.7864	15.2432
MB_Data_Usg_M07	0	13,672	94.2740	259.8391	17.6345
MB_Data_Usg_M08	0	16,297	109.5912	348.7336	16.9031
avg_days_susp	0	62	3.4714	3.8313	1.5937
bill_data_usg_m03	-13,678	40,767.1000	1,864.9142	1,634.5099	1.3974

5. Restore the view of the Interval Variable Moments table.

6. Expand the **Important Inputs** chart. Notice that the same **MB_Data_Usg_M** variables have been selected as being important variables. Only three of the six **MB_Data_Usg_M** variables are listed on the Interval Variable Moments table because, by default, only the variables found to be important are summarized in the results of the Data Exploration node. Importance is defined by a decision tree using PROC TREESPLIT. You should transform these three inputs.



Note: The entire chart is not shown above.

7. Restore the view of the Important Variables table.

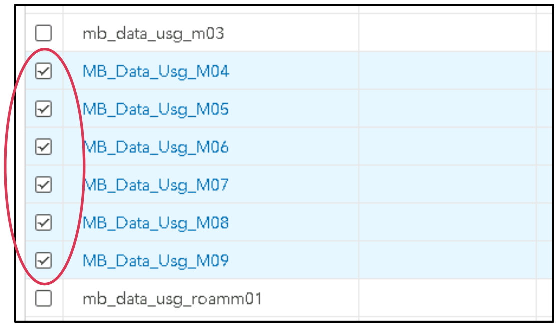
8. Close the Results window. Transformation rules are assigned on the Data tab. (Alternatively, transformation rules could be assigned using a Manage Variables node. In the Manage Variables

window, the **New Transform** column is hidden by default but could be displayed. Click the **Options** button in the upper right corner of the Manager Variables table, next to the New Order column. Then select **Manage Columns**. Recall, however, that rules established in the Manage Variables node are not saved if the pipeline is saved to the exchange.)

9. Click the **Data** tab. Make sure that any selected variables are deselected. It might help to sort by the **Variable Name** column if that column is currently not sorted.

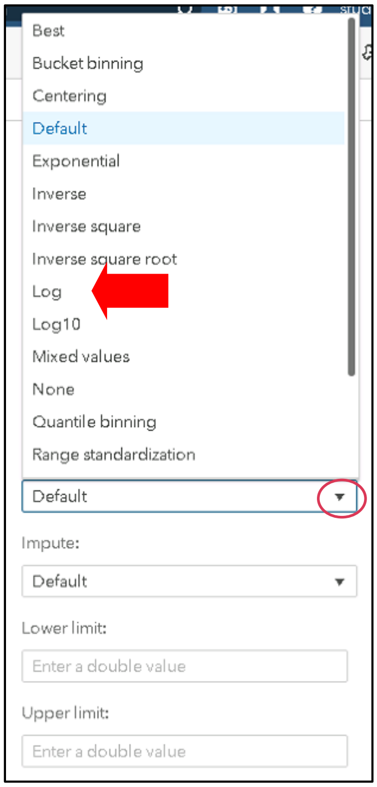
10. Scroll down until you see the **MB_Data_Usg_M** variables. Although only three were deemed as important in the Data Exploration node, let's apply a Log transformation to all six of them.

11. Select all six **MB_Data_Usg_M** variables by selecting the check boxes next to their names.



□ mb_data_usg_m03
☑ MB_Data_Usg_M04
☑ MB_Data_Usg_M05
☑ MB_Data_Usg_M06
☑ MB_Data_Usg_M07
☑ MB_Data_Usg_M08
☑ MB_Data_Usg_M09
□ mb_data_usg_roamm01

12. In the Multiple Variables window, in the right pane select the **Transform** menu and select **Log**.



Default ▾

Impute:

Default ▾

Lower limit:

Enter a double value

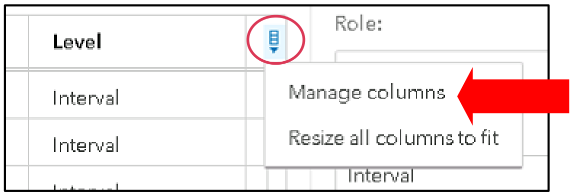
Upper limit:

Enter a double value

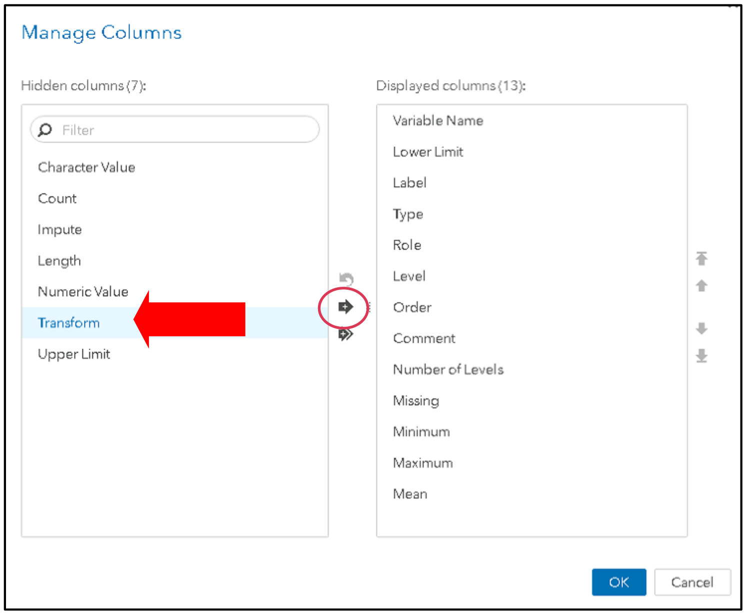
Note: Metadata always overrules transformations that are defined within the Transformation node. Metadata can be set using the Manage Variable node or on the Data tab.

The **Transform** column does not appear by default in the table on the Data tab, but it can be added.

13. Click the **Options** button in the upper right corner of the data table. Then select **Manage columns**.



14. In the Manage Columns window, under **Hidden columns**, select **Transform**, and then click the single right arrow that has a plus sign on it.



Manage Columns

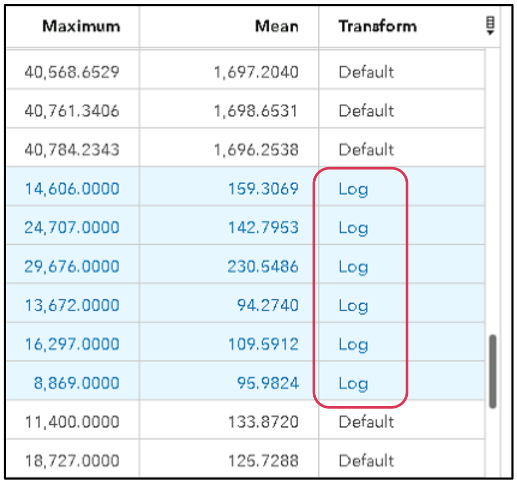
Hidden columns (7):

- Character Value
- Count
- Impute
- Length
- Numeric Value
- Transform
- Upper Limit

Displayed columns (13):

- Variable Name
- Lower Limit
- Label
- Type
- Role
- Level
- Order
- Comment
- Number of Levels
- Missing
- Minimum
- Maximum
- Mean

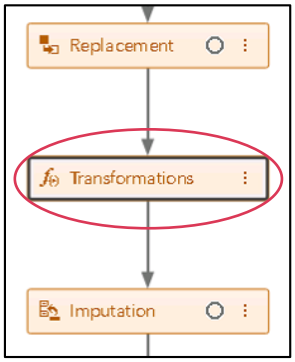
15. Click **OK**. On the Data tab, the Transform column can be seen by scrolling to the right. All six **MB_Data_Usg_M** variables show Log as the transformation rule.



Maximum	Mean	Transform
40,568.6529	1,697.2040	Default
40,761.3406	1,698.6531	Default
40,784.2343	1,696.2538	Default
14,606.0000	159.3069	Log
24,707.0000	142.7953	Log
29,676.0000	230.5486	Log
13,672.0000	94.2740	Log
16,297.0000	109.5912	Log
8,869.0000	95.9824	Log
11,400.0000	133.8720	Default
18,727.0000	125.7288	Default

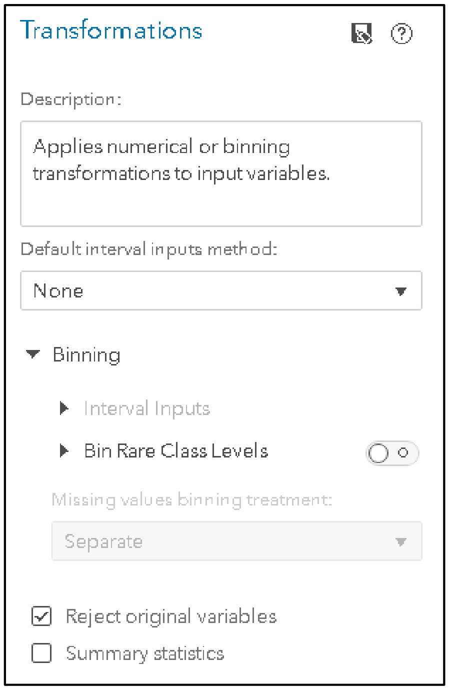
16. Return to the Starter Template pipeline.

17. Expand the left pane on **Nodes** if it is not opened. Under Data Mining Preprocessing, select the **Transformations** node and place it *between* the Replacement node and the Imputation node.



Note: The idea is you first change metadata in the Data tab or using the Manage Variables node to specify what you want to do with the variables (so far, you have seen Replacement and Transformation.) Then you need to add a node (in our example Replacement or Transformation) to make those changes to the data. The subsequent node (Replacement or Transformation) actually performs the changes you encoded in metadata.

18. Do ***not*** make changes to the properties of the Transformations node. Although the **Default interval inputs method** property indicates **None**, the metadata rules assigned to the variables under the Data tab override this default setting.


Transformations

Description:

Applies numerical or binning transformations to input variables.

Default interval inputs method:

None

Missing values binning treatment:

Separate

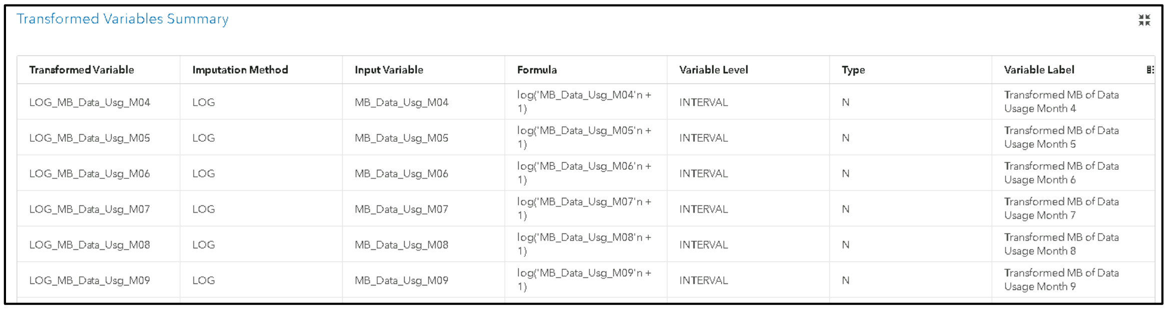
☑ Reject original variables

☐ Summary statistics

19. Right-click the **Transformation** node and select **Run**.

20. When the run is finished, open the results. Expand the **Transformed Variables Summary** table. This table displays information about the transformed variables, including how they were transformed, the corresponding input variable, the formula applied, the variable level, type, and variable label.

Notice that a Log transformation has been applied to all six **MB_Data_Usg_M** variables and that the term **LOG_** now appears at the beginning of the names of those variables.

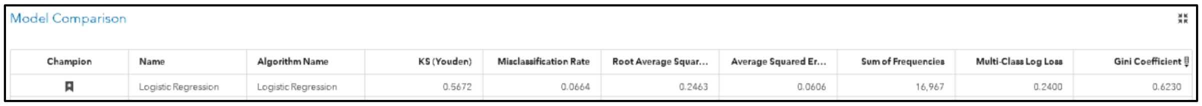


Transformed Variables Summary

Transformed Variable	Imputation Method	Input Variable	Formula	Variable Level	Type	Variable Label
LOG_MB_Data_Usg_M04	LOG	MB_Data_Usg_M04	log('MB_Data_Usg_M04'n + 1)	INTERVAL	N	Transformed MB of Data Usage Month 4
LOG_MB_Data_Usg_M05	LOG	MB_Data_Usg_M05	log('MB_Data_Usg_M05'n + 1)	INTERVAL	N	Transformed MB of Data Usage Month 5
LOG_MB_Data_Usg_M06	LOG	MB_Data_Usg_M06	log('MB_Data_Usg_M06'n + 1)	INTERVAL	N	Transformed MB of Data Usage Month 6
LOG_MB_Data_Usg_M07	LOG	MB_Data_Usg_M07	log('MB_Data_Usg_M07'n + 1)	INTERVAL	N	Transformed MB of Data Usage Month 7
LOG_MB_Data_Usg_M08	LOG	MB_Data_Usg_M08	log('MB_Data_Usg_M08'n + 1)	INTERVAL	N	Transformed MB of Data Usage Month 8
LOG_MB_Data_Usg_M09	LOG	MB_Data_Usg_M09	log('MB_Data_Usg_M09'n + 1)	INTERVAL	N	Transformed MB of Data Usage Month 9

Note: In the Formula column, notice that the formula for the Log transformations includes an offset of 1 to avoid the case of Log(0).

21. Restore the Transformed Variables Summary window and close the results.
22. Run the entire pipeline.
23. Open the results of the Model Comparison node.



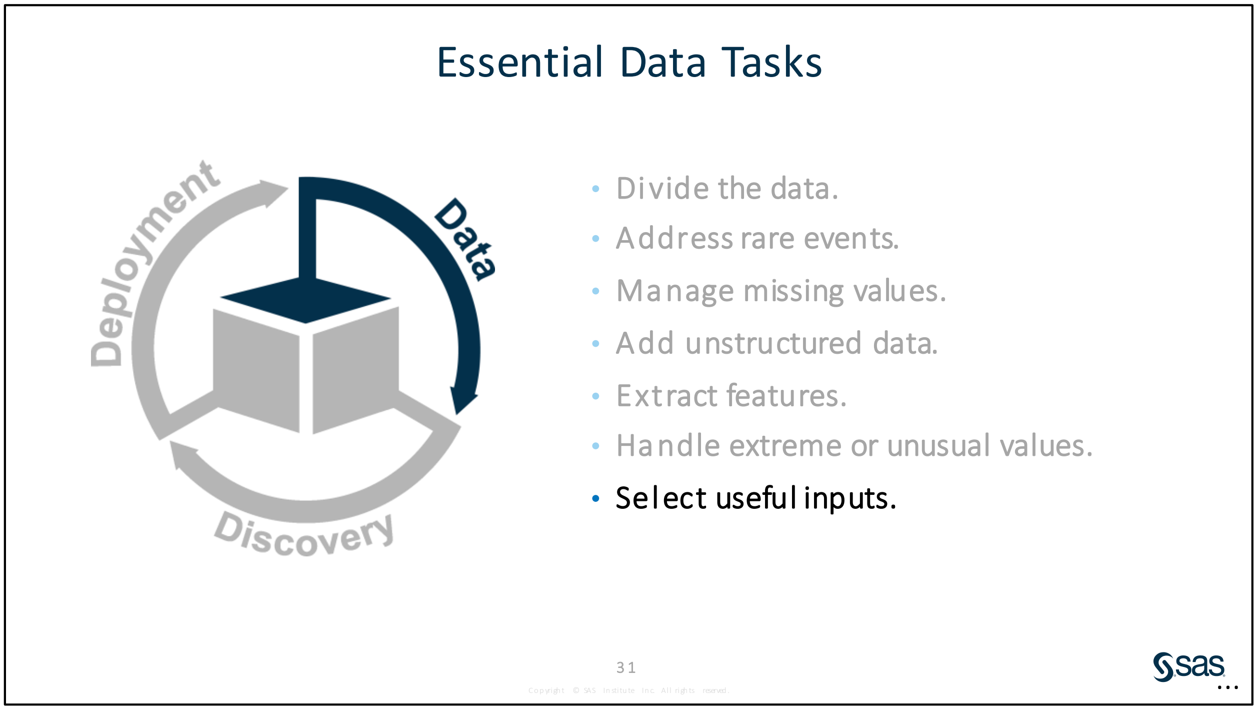
Model Comparison

Champion	Name	Algorithm Name	KS (Youden)	Misclassification Rate	Root Average Squar...	Average Squared Er...	Sum of Frequencies	Multi-Class Log Loss	Gini Coefficient
🔖	Logistic Regression	Logistic Regression	0.5672	0.0664	0.2463	0.0606	16,967	0.2400	0.6230

24. Close the Results window.

End of Demonstration

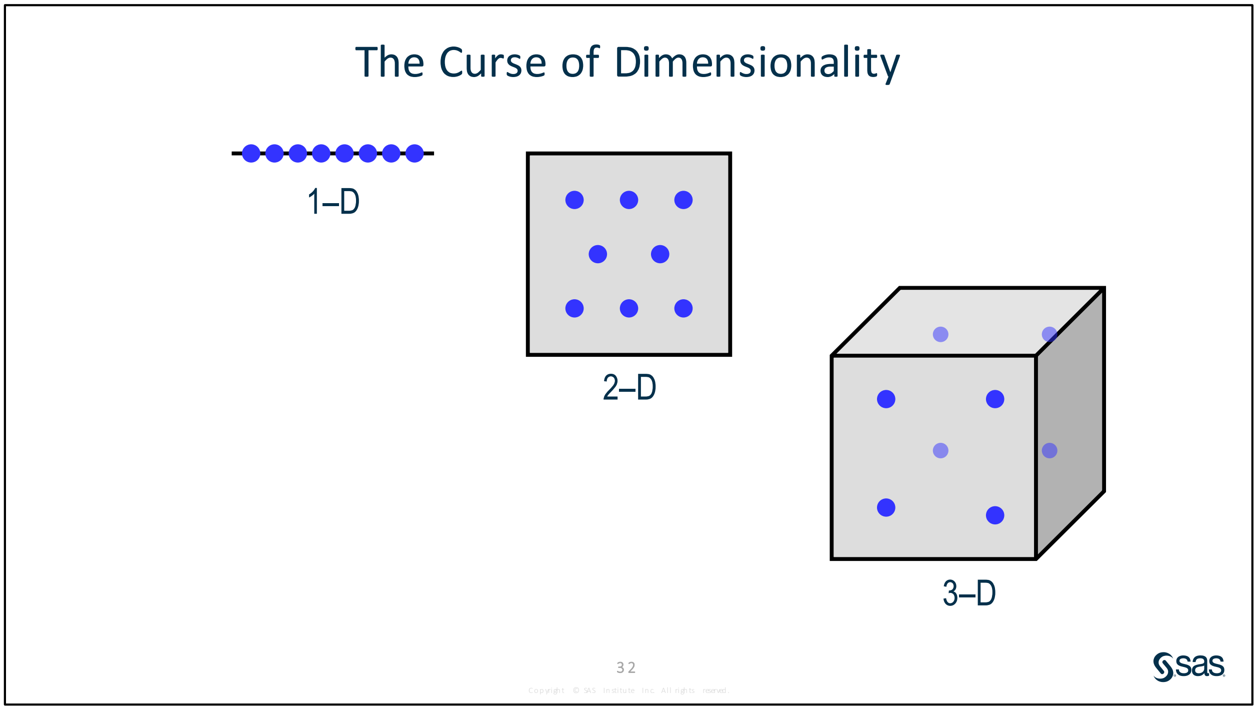
2.4 Feature Selection



Essential Data Tasks

- Divide the data.
- Address rare events.
- Manage missing values.
- Add unstructured data.
- Extract features.
- Handle extreme or unusual values.
- Select useful inputs.

31



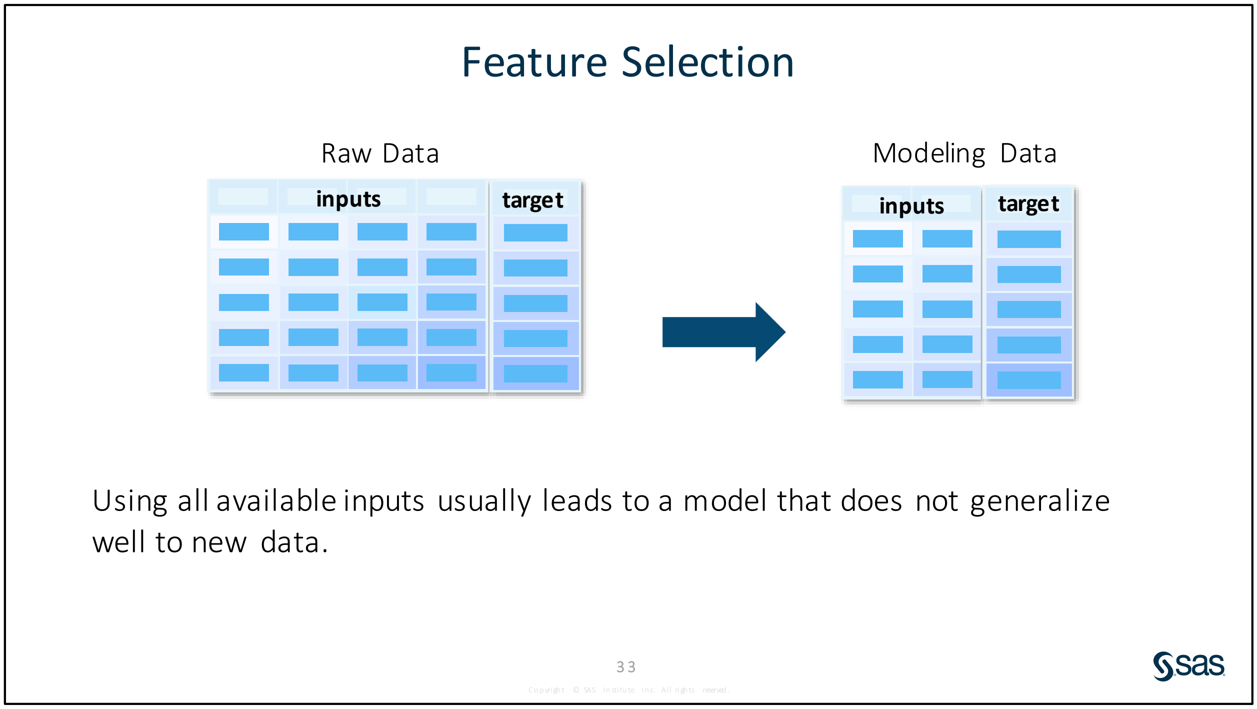
The Curse of Dimensionality

32

The *dimension* of a problem refers to the number of input variables (more accurately, *degrees of freedom*) that are available for creating a prediction. Data mining problems are often massive in dimension.

The *curse of dimensionality* refers to the exponential increase in data required to densely populate space as the dimension increases. For example, the eight points fill the one-dimensional space but become more separated as the dimension increases. In a 100-dimensional space, they would be like distant galaxies.

The curse of dimensionality limits your practical ability to fit a flexible model to noisy data (real data) when there are many input variables. A densely populated input space is required to fit highly complex models. When you assess how much data is available for data mining, you must consider the dimension of the problem.



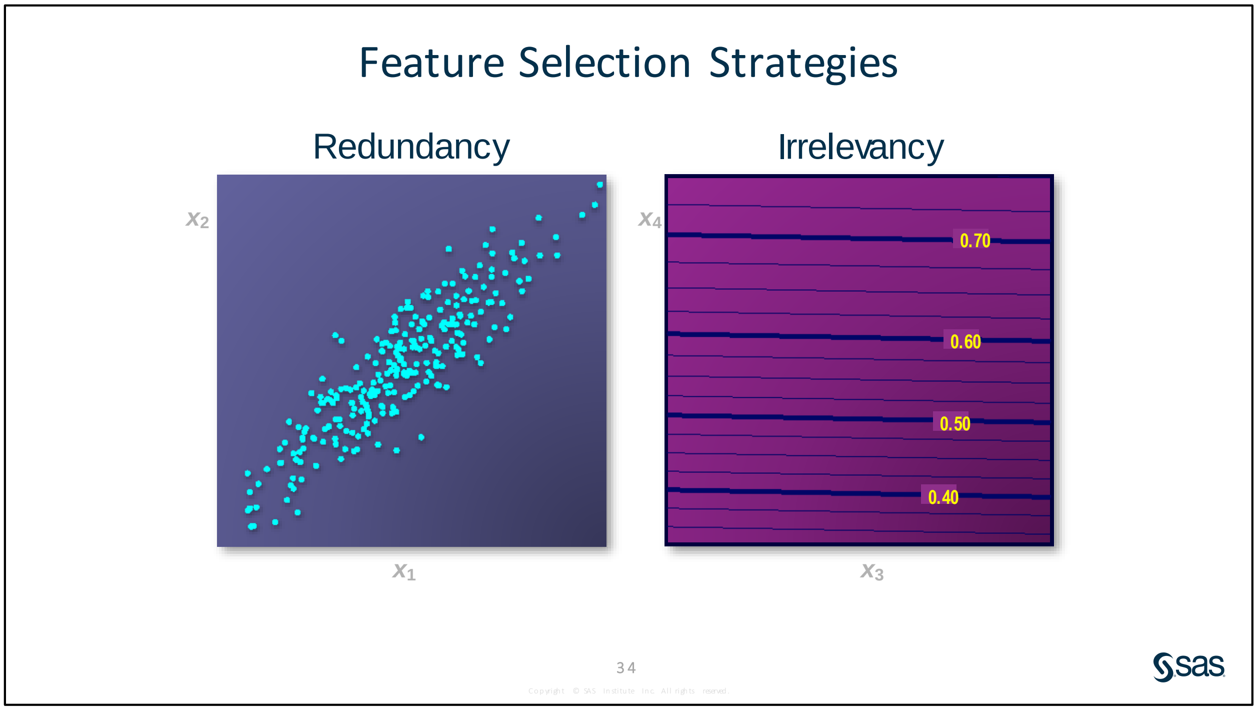
Feature Selection

Using all available inputs usually leads to a model that does not generalize well to new data.

33

When a model is fit using all available inputs, it typically results in a model that does not generalize well. The purpose of all predictive models is that they are eventually applied to new data. When a model is overfit to one data source, it might be very accurate at making predictions for that same data source, but it might lose a significant amount of accuracy when applied to new data. A model using all available inputs will likely be overfit to the data set used to construct the model. One way to avoid this is to use only a subset of all inputs in the final model.

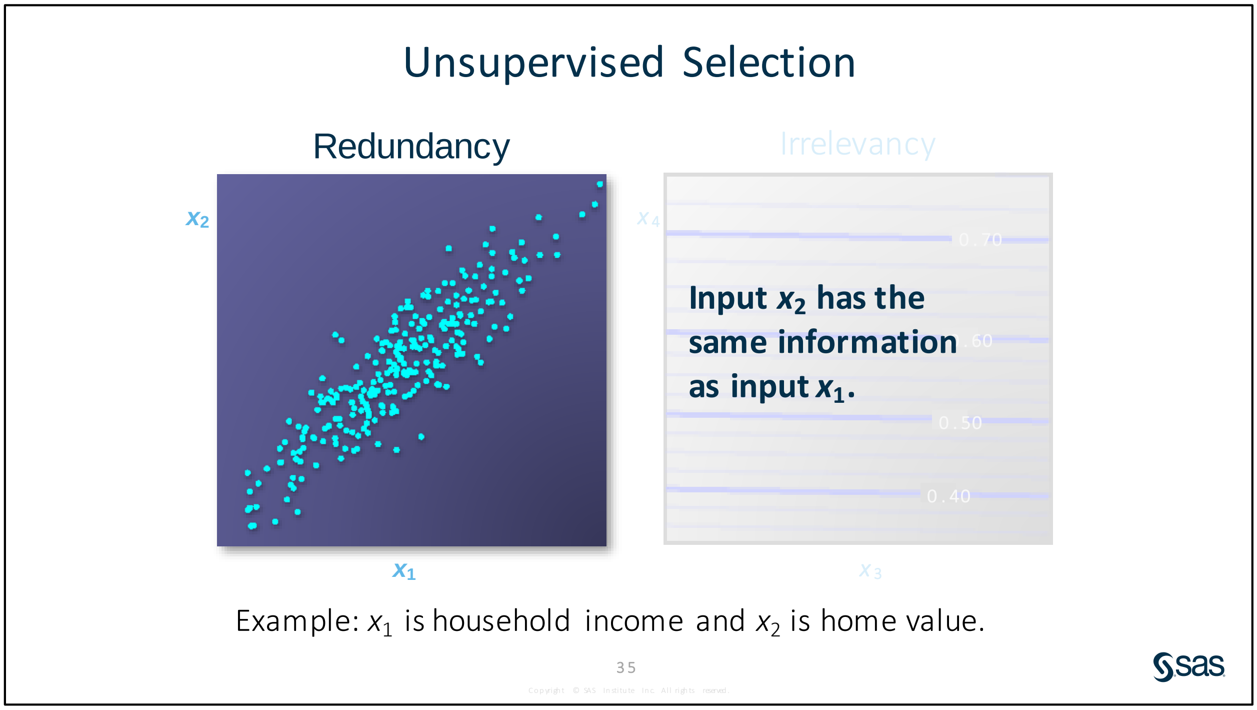
There are many techniques available for selecting inputs for a model. Some of these methods might be supervised, where the target variable is used in the process. Other techniques are unsupervised and ignore the target. Further, some modeling algorithms themselves might reduce the number of inputs during the model building process (for example, decision trees) but others might not (for example, neural networks), where some external method is used to select inputs.



Feature Selection Strategies

Input selection (that is, reducing the number of inputs) is the obvious way to thwart the curse of dimensionality. Unfortunately, reducing the dimension is also an easy way to disregard important information.

The two principal reasons for eliminating a variable are redundancy and irrelevancy.



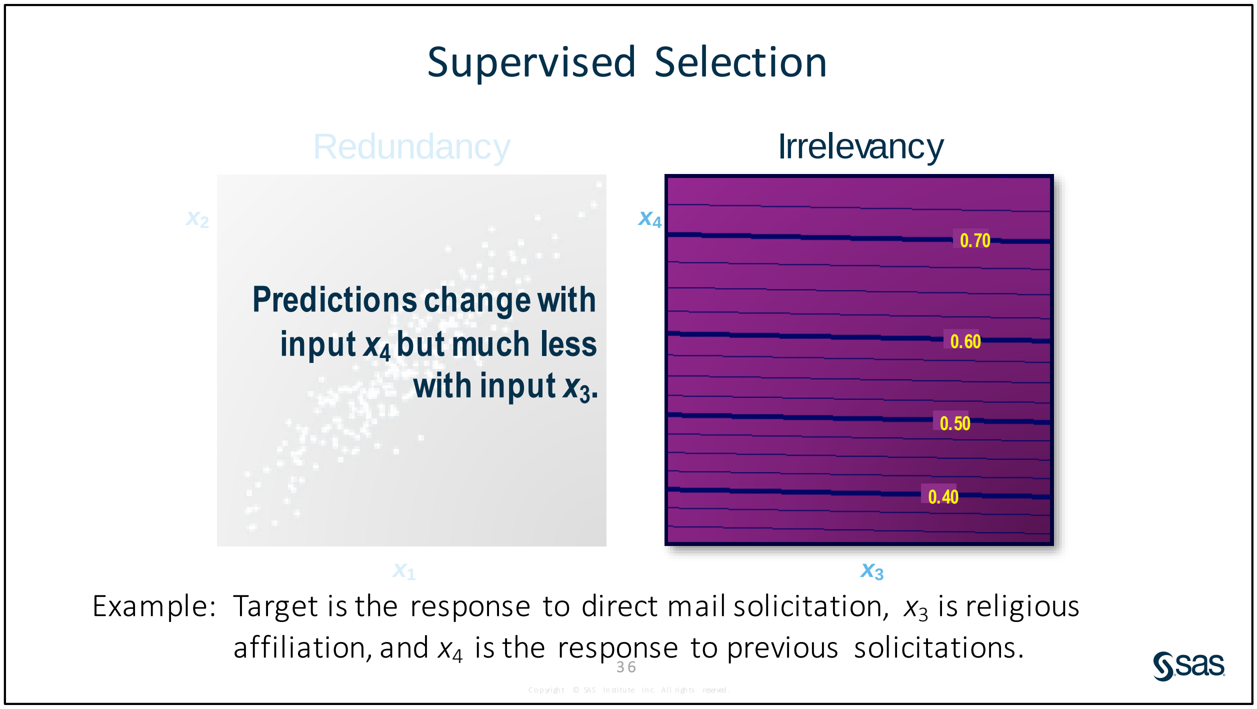
Unsupervised Selection

Input x_2 has the same information as input x_1.

Example: x_1 is household income and x_2 is home value.

A *redundant* input does not give any new information that was not already explained by other inputs. In the example above, knowing the value of input *x*1gives you a good idea of the value of *x*2.

For decision tree models, the modeling algorithm makes input redundancy a relatively minor issue. For other modeling tools, input redundancy requires more elaborate methods to mitigate the problem.



Supervised Selection

Redundancy

Irrelevancy

Predictions change with input x_4 but much less with input x_3.

Example: Target is the response to direct mail solicitation, x_3 is religious affiliation, and x_4 is the response to previous solicitations.

36

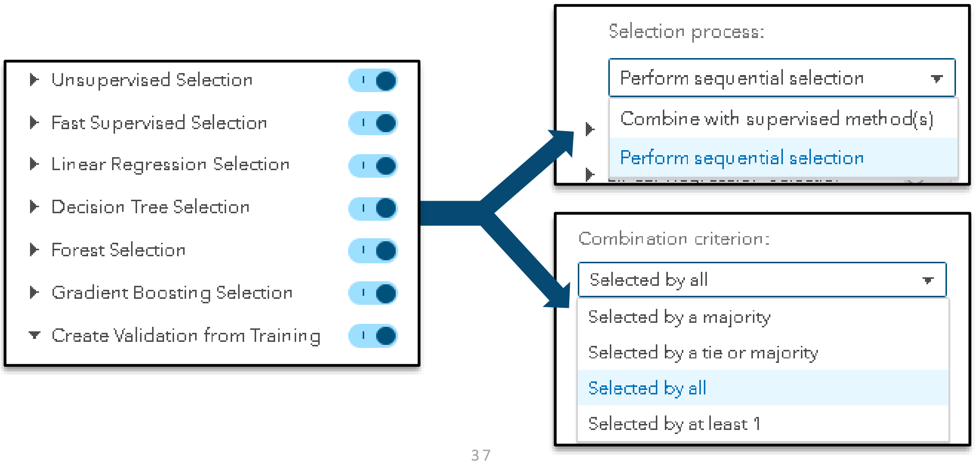
Copyright © SAS Institute Inc. All rights reserved.

An *irrelevant* input does not provide information about the target. In the example above, predictions change with input *x*4, but not with input *x*3.

For decision tree models, the modeling algorithm automatically ignores irrelevant inputs. Other modeling methods must be modified or rely on additional tools to properly deal with irrelevant inputs.

Feature Selection in Model Studio

The Variable Selection node performs unsupervised and several supervised methods of variable selection to reduce the number of inputs.



- Unsupervised Selection
- Fast Supervised Selection
- Linear Regression Selection
- Decision Tree Selection
- Forest Selection
- Gradient Boosting Selection
- Create Validation from Training

Selection process: Perform sequential selection

- Combine with supervised method(s)
- Perform sequential selection

Combination criterion: Selected by all

- Selected by a majority
- Selected by a tie or majority
- Selected by all
- Selected by at least 1

Many data mining databases have hundreds of potential model inputs (independent or explanatory variables) that can be used to predict the target (dependent or response variable). The Variable Selection node assists you in reducing the number of inputs by rejecting input variables based on the selection results. This node finds and selects the best variables for analysis by using unsupervised and supervised selection methods. You can choose among one or more of the available selection methods in the variable selection process.

If you choose the unsupervised selection method, you can specify in the **Selection process** property whether this method is run prior to the supervised methods (sequential selection). If you choose to perform a sequential selection, which is default, any variable rejected by the unsupervised method is not used by the subsequent supervised methods. If you are not performing a sequential selection, the results from the unsupervised method are combined with the chosen supervised methods.

If you choose multiple methods, the results from the individual methods are combined to generate the final selection result. This is done with *combination criterion*. This is a "voting" method such that each selection method gets a vote on whether a variable is selected. As an option, you choose at what voting level (combination criterion) a variable is selected. Voting levels range from the least restrictive option (at least one chosen method selects the variable) to the most restrictive option (all chosen methods select the variable). Any variable that is not selected in the final outcome is rejected, and subsequent nodes in the pipeline do not use that variable.

You also have the option to accomplish *pre-screening* of the input variables before running the chosen variable selection methods. In pre-screening, if a variable exceeds the maximum number of class levels threshold or the maximum missing percent threshold, that variable is rejected and not processed by the subsequent variable selection methods.

Note: The Advisor options also accomplishes variable pre-screening when the project is created, so this option can be used to increase the level of pre-screening over what is done at the project level.

Details: Variable Selection Methods

The following variable selection methods are available in the Variable Selection node:

- *Unsupervised Selection*: Identifies the set of input variables that jointly explains the maximum amount of data variance. The target variable is not considered with this method. Unsupervised Selection specifies the VARREDUCE procedure to perform unsupervised variable selection by identifying a set of variables that jointly explain the maximum amount of data variance. Variable selection is based on covariance analysis.
- *Fast Supervised Selection*: Identifies the set of input variables that jointly explain the maximum amount of variance contained in the target. Fast Supervised Selection specifies the VARREDUCE procedure to perform supervised variable selection by identifying a set of variables that jointly explain the maximum amount of variance contained in the response variables. Supervised selection is essentially based on AIC, AICC, and BIC stop criterion.
- *Linear Regression Selection*: Fits and performs variable selection on an ordinary least squares regression predictive model. This is valid for an interval target and a binary target. In the case of a character binary target (or a binary target with a user-defined format), a temporary numeric variable with values of 0 or 1 is created, which is then substituted for the target. Linear Regression Selection specifies the REGSELECT procedure to perform linear regression selection based on ordinary least square regression. It offers many effect-selection methods, including Backward, Forward, Forward-swap, Stepwise methods, and modern LASSO and Adaptive LASSO methods. It also offers extensive capabilities for customizing the model selection by using a wide variety of selection and stopping criteria, from computationally efficient significance level-based criteria to modern, computationally intensive validation-based criteria.
- *Decision Tree Selection*: Trains a decision tree predictive model. The residual sum of squares variable importance is calculated for each predictor variable, and the relative variable importance threshold that you specify is used to select the most useful predictor variables. Decision Tree Selection specifies the TREESPLIT procedure to perform decision tree selection based on CHAID, Chi-square, Entropy, Gini, Information gain ratio, F test, and Variance target criterion. It produces a classification tree, which models a categorical response, or a regression tree, which models a continuous response. Both types of trees are called decision trees because the model is expressed as a series of IF-THEN statements.
- *Forest Selection*: Trains a forest predictive model by fitting multiple decision trees. The residual sum of squares variable importance is calculated for each predictor variable, averaged across all the trees, and the relative variable importance threshold that you specify is used to select the most useful predictor variables. Forest Selection specifies the FOREST procedure to create a predictive model that consists of multiple decision trees.
- *Gradient Boosting Selection*: Trains a gradient boosting predictive model by fitting a set of additive decision trees. The residual sum of squares variable importance is calculated for each predictor variable, averaged across all the trees, and the relative variable importance threshold that you specify is used to select the most useful predictor variables. Gradient Boosting Selection specifies the GRADBOOST procedure to create a predictive model that consists of multiple decision trees.

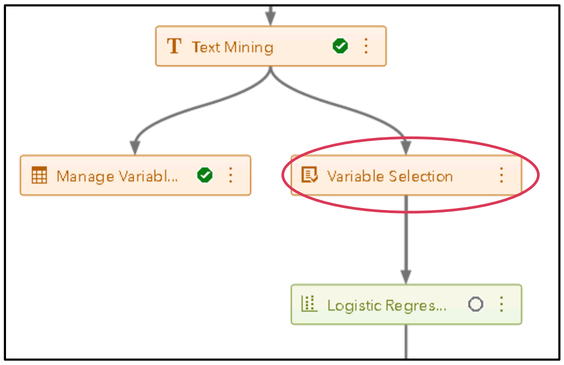
- *Create Validation Sample from Training Data* specifies whether a validation sample should be created from the incoming training data. This is recommended even if the data have already been partitioned so that only the training partition is used for variable selection, and the validation partition can be used for modeling.

Note: You can choose all the above methods for variable selection, but this might result in enormous processing time depending on your data and distributed environment. Another important thing to note is that if you use the Text Mining node before the Variable Selection node in the pipeline, and any of the tree-based methods (decision trees, forest and gradient boosting) are used for variable selection in addition to the default methods, an error is returned. The log indicates that the ID for tree appears multiple times in the same BY group. Decision trees, forest, and gradient boosting are discussed in the next chapter.

Selecting Features

In this demonstration, you use the Variable Selection node to reduce the number of inputs for modeling.

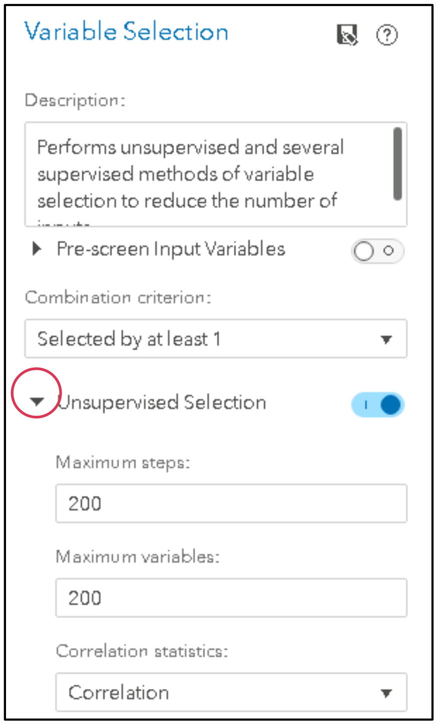
1. In the Starter Template, place a Variable Selection node ***between*** the Text Mining node and the Logistic Regression node.



2. Select the **Variable Selection** node.

   In the properties, varying combinations of criteria can be used to select inputs. Keep **Combination Criterion** at **Selected by at least 1**. This means that any input selected by at least one of the selection criteria chosen is passed on to subsequent nodes as inputs. The **Fast Supervised Selection** method is selected by default. The **Create Validation from Training** property is also selected by default, but its button is initially disabled.

3. In addition, turn on the **Unsupervised Selection** and **Linear Regression Selection** methods by clicking the button slider ○ ∘ next to each property name. When a property is turned on, additional options appear. The screen capture below shows the additional options of the **Unsupervised Selection** method after it is selected. You can hide the new options by selecting the down arrow next to the property name.



Variable Selection

Description:

Performs unsupervised and several supervised methods of variable selection to reduce the number of

▸ Pre-screen Input Variables ○ ∘

Combination criterion:

Selected by at least 1 ▾

▾ Unsupervised Selection ●

Maximum steps:

200

Maximum variables:

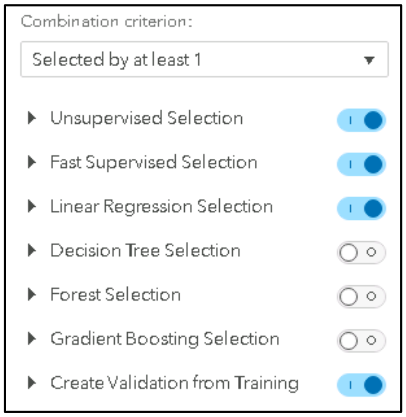
200

Correlation statistics:

Correlation ▾

Keep the default settings for all the new options that appear for the Unsupervised Selection and Linear Regression Selection methods.

After the **Unsupervised Selection** and **Linear Regression Selection** methods are selected and the options for each are hidden, the properties panel resembles the following:



Combination criterion:

Selected by at least 1 ▾

▸ Unsupervised Selection ●

▸ Fast Supervised Selection ●

▸ Linear Regression Selection ●

▸ Decision Tree Selection ○ ∘

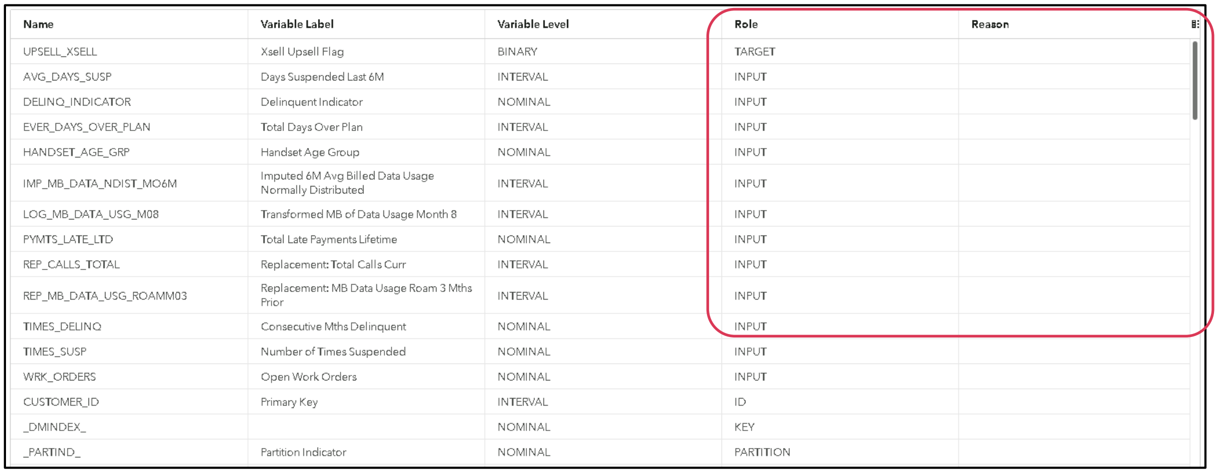
▸ Forest Selection ○ ∘

▸ Gradient Boosting Selection ○ ∘

▸ Create Validation from Training ●

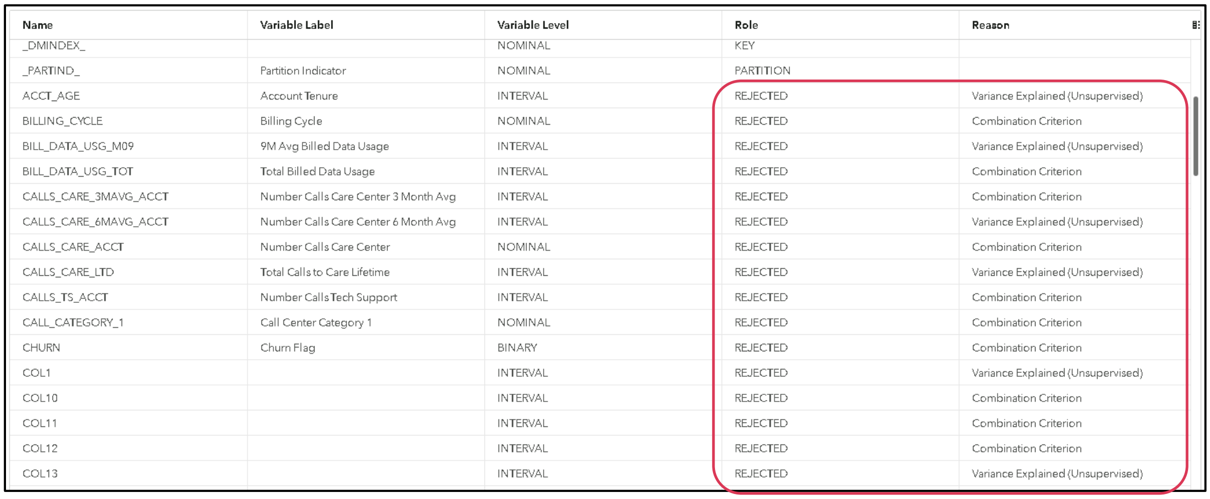
Note: The **Create Validation from Training** property was initially selected by default, but the slider button did not become active until another method was selected. This property specifies whether a validation sample should be created from the incoming training data. It is recommended to create this validation set even if the data has already been partitioned so that only the training partition is used for variable selection and the original validation partition can be used for modeling.

4. Run the **Variable Selection** node and view the results when it is complete.
5. Expand the **Variable Selection** table. This table contains the output role for each variable. At the top of the table are the input variables selected by the node. These variables have a blank cell in the **Reason** column.



Name	Variable Label	Variable Level	Role	Reason
UPSELL_XSELL	Xsell Upsell Flag	BINARY	TARGET	
AVG_DAYS_SUSP	Days Suspended Last 6M	INTERVAL	INPUT	
DELINQ_INDICATOR	Delinquent Indicator	NOMINAL	INPUT	
EVER_DAYS_OVER_PLAN	Total Days Over Plan	INTERVAL	INPUT	
HANDSET_AGE_GRP	Handset Age Group	NOMINAL	INPUT	
IMP_MB_DATA_NDIST_MO6M	Imputed 6M Avg Billed Data Usage Normally Distributed	INTERVAL	INPUT	
LOG_MB_DATA_USG_M08	Transformed MB of Data Usage Month 8	INTERVAL	INPUT	
PYMTS_LATE_LTD	Total Late Payments Lifetime	NOMINAL	INPUT	
REP_CALLS_TOTAL	Replacement: Total Calls Curr	INTERVAL	INPUT	
REP_MB_DATA_USG_ROAMM03	Replacement: MB Data Usage Roam 3 Mths Prior	INTERVAL	INPUT	
TIMES_DELINQ	Consecutive Mths Delinquent	NOMINAL	INPUT	
TIMES_SUSP	Number of Times Suspended	NOMINAL	INPUT	
WRK_ORDERS	Open Work Orders	NOMINAL	INPUT	
CUSTOMER_ID	Primary Key	INTERVAL	ID	
DMINDEX		NOMINAL	KEY	
PARTIND	Partition Indicator	NOMINAL	PARTITION	

6. Scroll down in the Variable Selection table. It shows which variables have been rejected by the node. The reason for rejection is shown in the **Reason** column. Only a subset of the rejected variables is shown below.



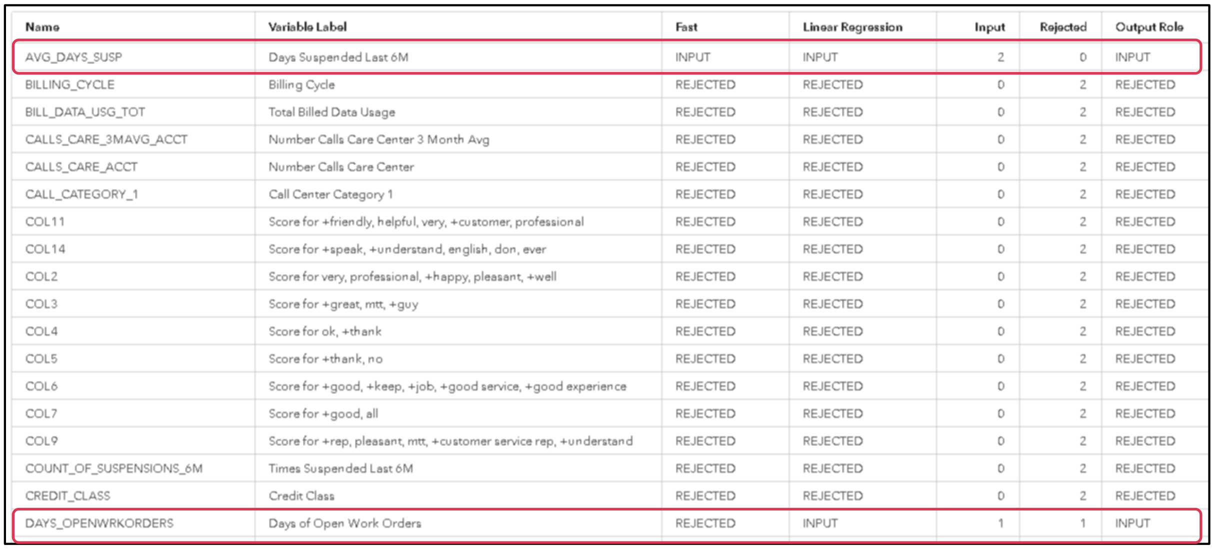
Name	Variable Label	Variable Level	Role	Reason
DMINDEX		NOMINAL	KEY	
PARTIND	Partition Indicator	NOMINAL	PARTITION	
ACCT_AGE	Account Tenure	INTERVAL	REJECTED	Variance Explained (Unsupervised)
BILLING_CYCLE	Billing Cycle	NOMINAL	REJECTED	Combination Criterion
BILL_DATA_USG_M09	9M Avg Billed Data Usage	INTERVAL	REJECTED	Variance Explained (Unsupervised)
BILL_DATA_USG_TOT	Total Billed Data Usage	INTERVAL	REJECTED	Combination Criterion
CALLS_CARE_3MAVG_ACCT	Number Calls Care Center 3 Month Avg	INTERVAL	REJECTED	Combination Criterion
CALLS_CARE_6MAVG_ACCT	Number Calls Care Center 6 Month Avg	INTERVAL	REJECTED	Variance Explained (Unsupervised)
CALLS_CARE_ACCT	Number Calls Care Center	NOMINAL	REJECTED	Combination Criterion
CALLS_CARE_LTD	Total Calls to Care Lifetime	INTERVAL	REJECTED	Variance Explained (Unsupervised)
CALLS_TS_ACCT	Number Calls Tech Support	INTERVAL	REJECTED	Combination Criterion
CALL_CATEGORY_1	Call Center Category 1	NOMINAL	REJECTED	Combination Criterion
CHURN	Churn Flag	BINARY	REJECTED	Combination Criterion
COL1		INTERVAL	REJECTED	Variance Explained (Unsupervised)
COL10		INTERVAL	REJECTED	Combination Criterion
COL11		INTERVAL	REJECTED	Combination Criterion
COL12		INTERVAL	REJECTED	Combination Criterion
COL13		INTERVAL	REJECTED	Variance Explained (Unsupervised)

The Variable Selection table shows the variables that are rejected because of the variable selection and pre-screening process (turned-off in this case), as well as the reason for the rejection. This is in addition to other variables not processed by the Variable Selection node.

Recall that sequential selection (default) is performed, and any variable rejected by the unsupervised method is not used by the subsequent supervised methods. The variables that are rejected by supervised methods are represented by combination criterion (at least one in this case) in the **Reason** column. If you want to see whether they were selected or rejected by each method, look the Variable Selection Combination Summary.

7. Restore the view of the Variable Selection table and close the results.

8. Expand the **Variable Selection Combination Summary** table. For each variable that it includes the result (Input or Rejected) for each method that was used, the total count of each result, and the final output role (Input or Rejected). Finally, specific output for each selection method is also available in the results. Only a subset of the variables is shown below.



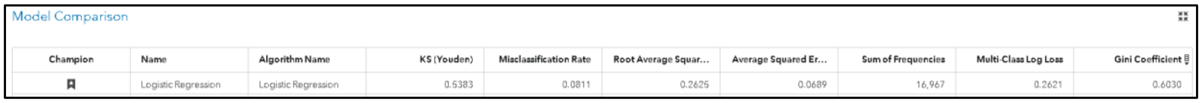
Name	Variable Label	Fast	Linear Regression	Input	Rejected	Output Role
AVG_DAYS_SUSP	Days Suspended Last 6M	INPUT	INPUT	2	0	INPUT
BILLING_CYCLE	Billing Cycle	REJECTED	REJECTED	0	2	REJECTED
BILL_DATA_USG_TOT	Total Billed Data Usage	REJECTED	REJECTED	0	2	REJECTED
CALLS_CARE_3MAVG_ACCT	Number Calls Care Center 3 Month Avg	REJECTED	REJECTED	0	2	REJECTED
CALLS_CARE_ACCT	Number Calls Care Center	REJECTED	REJECTED	0	2	REJECTED
CALL_CATEGORY_1	Call Center Category 1	REJECTED	REJECTED	0	2	REJECTED
COL11	Score for +friendly, helpful, very, +customer, professional	REJECTED	REJECTED	0	2	REJECTED
COL14	Score for +speak, +understand, english, don, ever	REJECTED	REJECTED	0	2	REJECTED
COL2	Score for very, professional, +happy, pleasant, +well	REJECTED	REJECTED	0	2	REJECTED
COL3	Score for +great, mtt, +guy	REJECTED	REJECTED	0	2	REJECTED
COL4	Score for ok, +thank	REJECTED	REJECTED	0	2	REJECTED
COL5	Score for +thank, no	REJECTED	REJECTED	0	2	REJECTED
COL6	Score for +good, +keep, +job, +good service, +good experience	REJECTED	REJECTED	0	2	REJECTED
COL7	Score for +good, all	REJECTED	REJECTED	0	2	REJECTED
COL9	Score for +rep, pleasant, mtt, +customer service rep, +understand	REJECTED	REJECTED	0	2	REJECTED
COUNT_OF_SUSPENSIONS_6M	Times Suspended Last 6M	REJECTED	REJECTED	0	2	REJECTED
CREDIT_CLASS	Credit Class	REJECTED	REJECTED	0	2	REJECTED
DAYS_OPENWRKORDERS	Days of Open Work Orders	REJECTED	INPUT	1	1	INPUT

The first variable in the table is selected by both fast-supervised selection and linear regression, the last variable in the table is selected by only the linear regression selection, and because the combination criterion is at least one, the output role of this variable is Input.

9. Restore the view of the Variable Selection Combination Summary table and close the results.

10. Run the pipeline by clicking the **Run pipeline** button.

11. Open the results of the Model Comparison node.



Model Comparison

Champion	Name	Algorithm Name	KS (Youden)	Misclassification Rate	Root Average Squar...	Average Squared Er...	Sum of Frequencies	Multi-Class Log Loss	Gini Coefficient
🔖	Logistic Regression	Logistic Regression	0.5383	0.0811	0.2625	0.0689	16,967	0.2621	0.6030

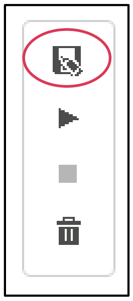
12. Close the results of the Model Comparison node.

End of Demonstration

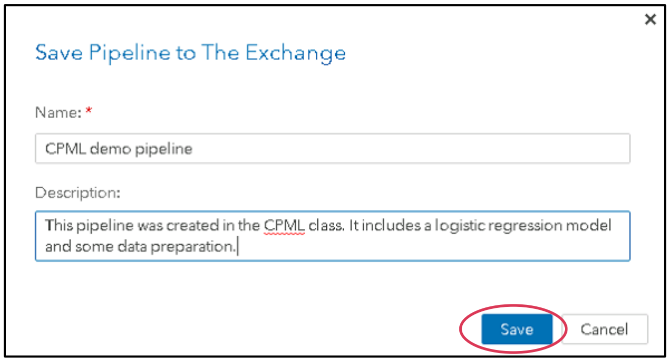
Saving a Pipeline to the Exchange

The current Starter Template pipeline is in multiple demonstrations of machine learning algorithms. In this demonstration, you save the Starter Template pipeline to the Exchange, where it will be available for other users.

1. In the upper right corner of the Starter Template pipeline window, click the **Save pipeline to The Exchange** button.



2. Change the name of the pipeline to **CPML demo pipeline**, and for the description, enter **This pipeline was created in the CPML class. It includes a logistic regression model and some data preparation.**. Click **Save**.



Save Pipeline to The Exchange

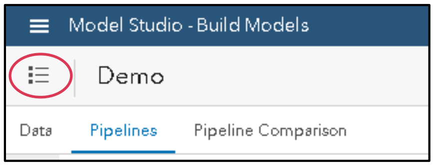
Name: *

CPML demo pipeline

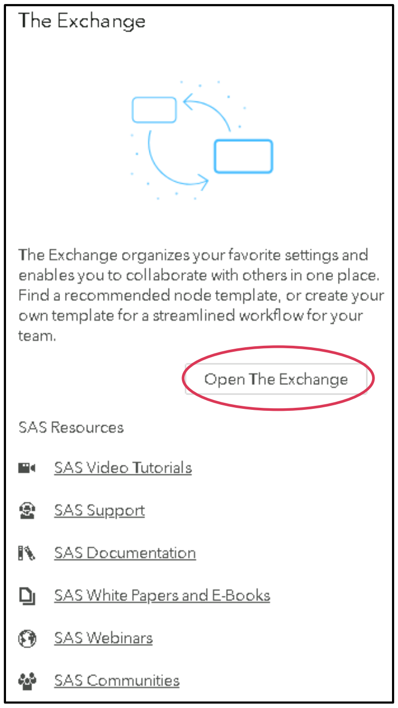
Description:

This pipeline was created in the CPML class. It includes a logistic regression model and some data preparation.

3. To see the saved pipeline in the Exchange, you must exit the current project. Click the **View all projects** button in the upper left corner to exit the Demo project.



4. Click **Open The Exchange** in the right side pane.



The Exchange

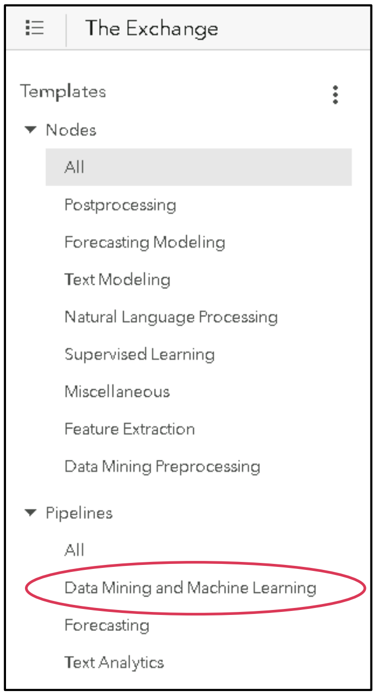
The Exchange organizes your favorite settings and enables you to collaborate with others in one place. Find a recommended node template, or create your own template for a streamlined workflow for your team.

Open The Exchange

SAS Resources

- SAS Video Tutorials
- SAS Support
- SAS Documentation
- SAS White Papers and E-Books
- SAS Webinars
- SAS Communities

5. In the left pane, expand **Pipelines** and select **Data Mining and Machine Learning**.

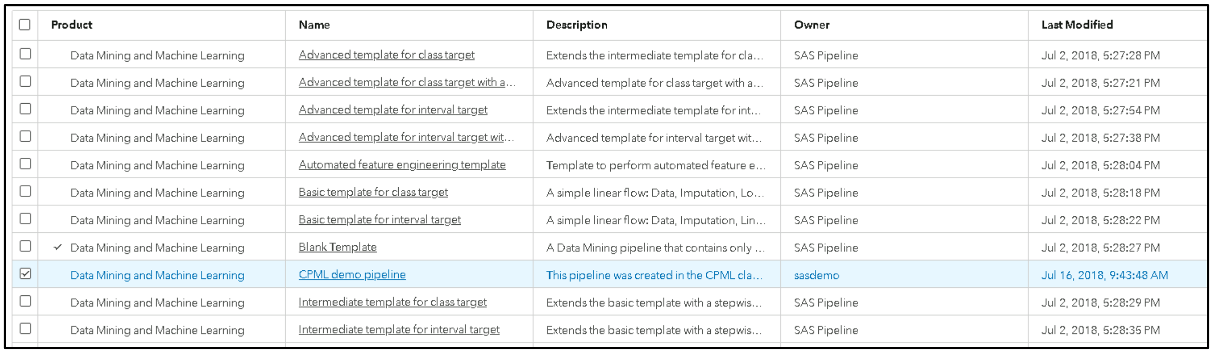


The Exchange

Templates

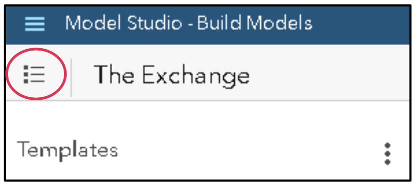
- Nodes
  - All
  - Postprocessing
  - Forecasting Modeling
  - Text Modeling
  - Natural Language Processing
  - Supervised Learning
  - Miscellaneous
  - Feature Extraction
  - Data Mining Preprocessing
- Pipelines
  - All
  - Data Mining and Machine Learning
  - Forecasting
  - Text Analytics

The newly saved CPML demo pipeline is added to the list of pipeline templates.



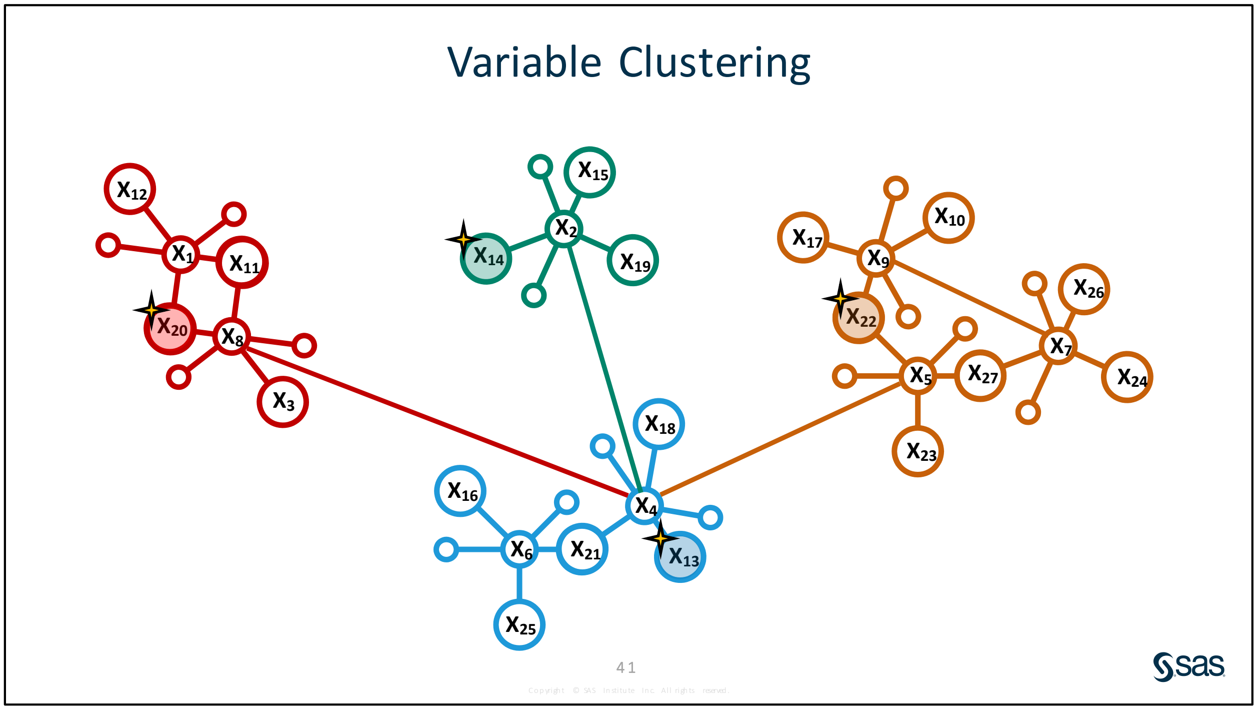
□	Product	Name	Description	Owner	Last Modified
□	Data Mining and Machine Learning	Advanced template for class target	Extends the intermediate template for cla…	SAS Pipeline	Jul 2, 2018, 5:27:28 PM
□	Data Mining and Machine Learning	Advanced template for class target with a…	Advanced template for class target with a…	SAS Pipeline	Jul 2, 2018, 5:27:21 PM
□	Data Mining and Machine Learning	Advanced template for interval target	Extends the intermediate template for int…	SAS Pipeline	Jul 2, 2018, 5:27:54 PM
□	Data Mining and Machine Learning	Advanced template for interval target wit…	Advanced template for interval target wit…	SAS Pipeline	Jul 2, 2018, 5:27:38 PM
□	Data Mining and Machine Learning	Automated feature engineering template	Template to perform automated feature e…	SAS Pipeline	Jul 2, 2018, 5:28:04 PM
□	Data Mining and Machine Learning	Basic template for class target	A simple linear flow: Data, Imputation, Lo…	SAS Pipeline	Jul 2, 2018, 5:28:18 PM
□	Data Mining and Machine Learning	Basic template for interval target	A simple linear flow: Data, Imputation, Lin…	SAS Pipeline	Jul 2, 2018, 5:28:22 PM
□	✓ Data Mining and Machine Learning	Blank Template	A Data Mining pipeline that contains only …	SAS Pipeline	Jul 2, 2018, 5:28:27 PM
☑	Data Mining and Machine Learning	CPML demo pipeline	This pipeline was created in the CPML cla…	sasdemo	Jul 16, 2018, 9:43:48 AM
□	Data Mining and Machine Learning	Intermediate template for class target	Extends the basic template with a stepwis…	SAS Pipeline	Jul 2, 2018, 5:28:29 PM
□	Data Mining and Machine Learning	Intermediate template for interval target	Extends the basic template with a stepwis…	SAS Pipeline	Jul 2, 2018, 5:28:35 PM

6. To exit the Exchange, click the **View all items** button in the upper left corner.



End of Demonstration

2.5 Variable Clustering (Self-Study)



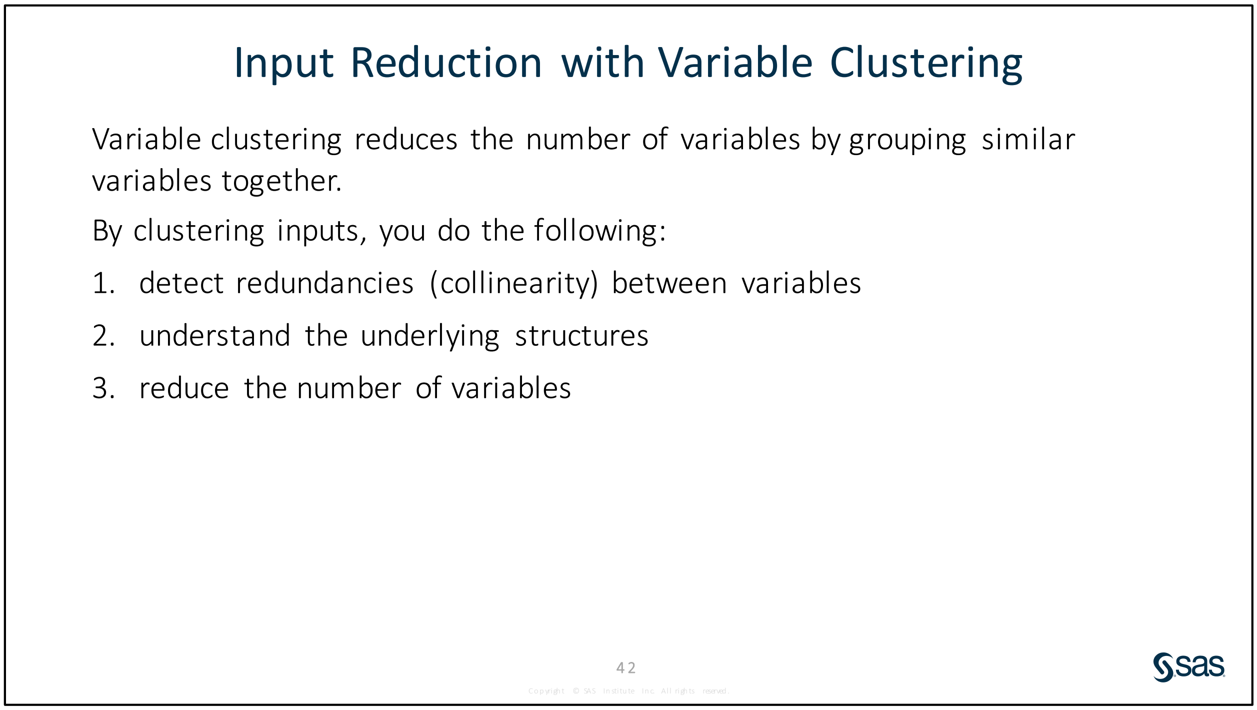
Variable Clustering

41

When presented with many variables to predict an outcome, you might want to reduce the number of variables in some way to make the prediction problem easier to tackle. Many of these variables are redundant, the concept that has already been introduced in Chapter 1. Including redundant inputs can degrade the analysis by

- destabilizing the parameter estimates
- increasing the risk of overfitting
- confounding interpretation
- increasing computation time
- increasing scoring effort
- increasing the cost of data collection and augmentation.

One approach to variable reduction is variable clustering. Variable clustering divides numeric variables into disjoint or hierarchical clusters. Variables in different clusters are conditionally independent given their own clusters. For each cluster that contains more than one variable, the variable that contributes the most to the variation in that cluster is chosen as the representative variable. All other variables are rejected.



Input Reduction with Variable Clustering

Variable clustering reduces the number of variables by grouping similar variables together.

By clustering inputs, you do the following:

1. detect redundancies (collinearity) between variables
2. understand the underlying structures
3. reduce the number of variables

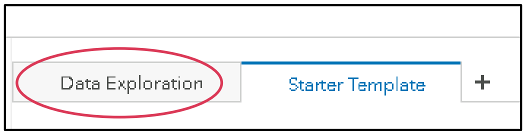
42

Variable clustering is a useful technique for data reduction because it finds the best variables for analysis. It removes collinearity, decreases variable redundancy, and helps reveal the underlying structure of the input variables in a data set in the sense that the groups of variables reveal the main dimensionalities of the data.

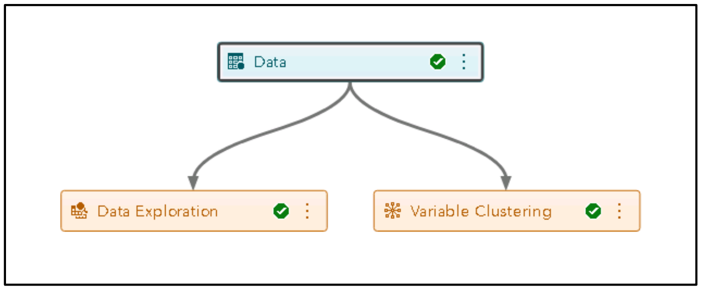
Clustering Inputs for Data Reduction (Self-Study)

In this demonstration, you use the Variable Clustering node to reduce the number of inputs for modeling.

1. Return to the Demo project and open the **Data Exploration** pipeline by clicking on its tab.

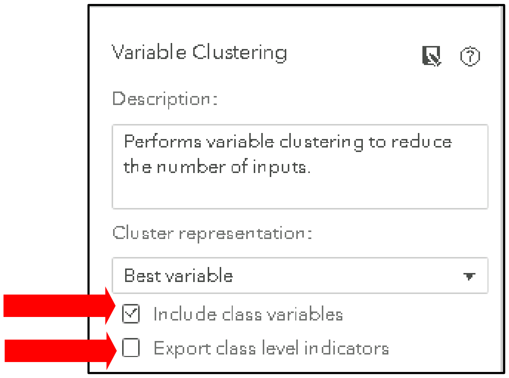


2. In the Data Exploration pipeline, right-click the **Data** node and select **Add below** ⇨ **Data Mining Preprocessing** ⇨ **Variable Clustering**. Your pipeline should resemble the following:



3. Select the **Variable Clustering** node. In the properties, you have an option of including categorical variables in the analysis. Turn the option on by selecting the **Include class variables** box. This means class variables are also used in variable clustering.

   Also, clear the box for **Export class level indicators**. This specifies not to export the class level indicators to replace the original class variables.



Variable Clustering

Description:

Performs variable clustering to reduce the number of inputs.

Cluster representation:

Best variable

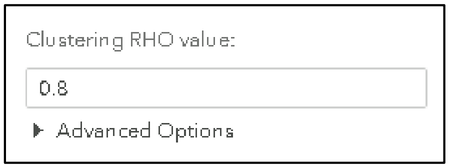
☑ Include class variables

☐ Export class level indicators

Note: Class variables are handled in a different way in Model Studio. Individual binary class level variables are used in the clustering process, but the original class variables are kept or dropped in the selection process. This selection depends on the variables that are included in a cluster, and the variable or variable level that is selected from each cluster.

Note: Also note that you have a Cluster representation property where you can choose to export the first principal component for each cluster (property value "Cluster component"). With the "Cluster component" option, the first principal component is extracted from all variables in a cluster and output as new variable _CLUSn (for example, _CLUS1, _CLUS2, _CLUS3, and so on), and the original cluster variables are rejected. The total number of generated component variables corresponds to the number of identified clusters. For more information, see "Three new Variable Clustering features in SAS Model Studio 8.3" at https://communities.sas.com/t5/SAS-Communities-Library/Three-new-Variable-Clustering-features-in-SAS-Model-Studio-8-3/ta-p/489430.

4. Observe that the default value of the regularization parameter Rho (ρ) is 0.8. You do not need to change its value.



Clustering RHO value:

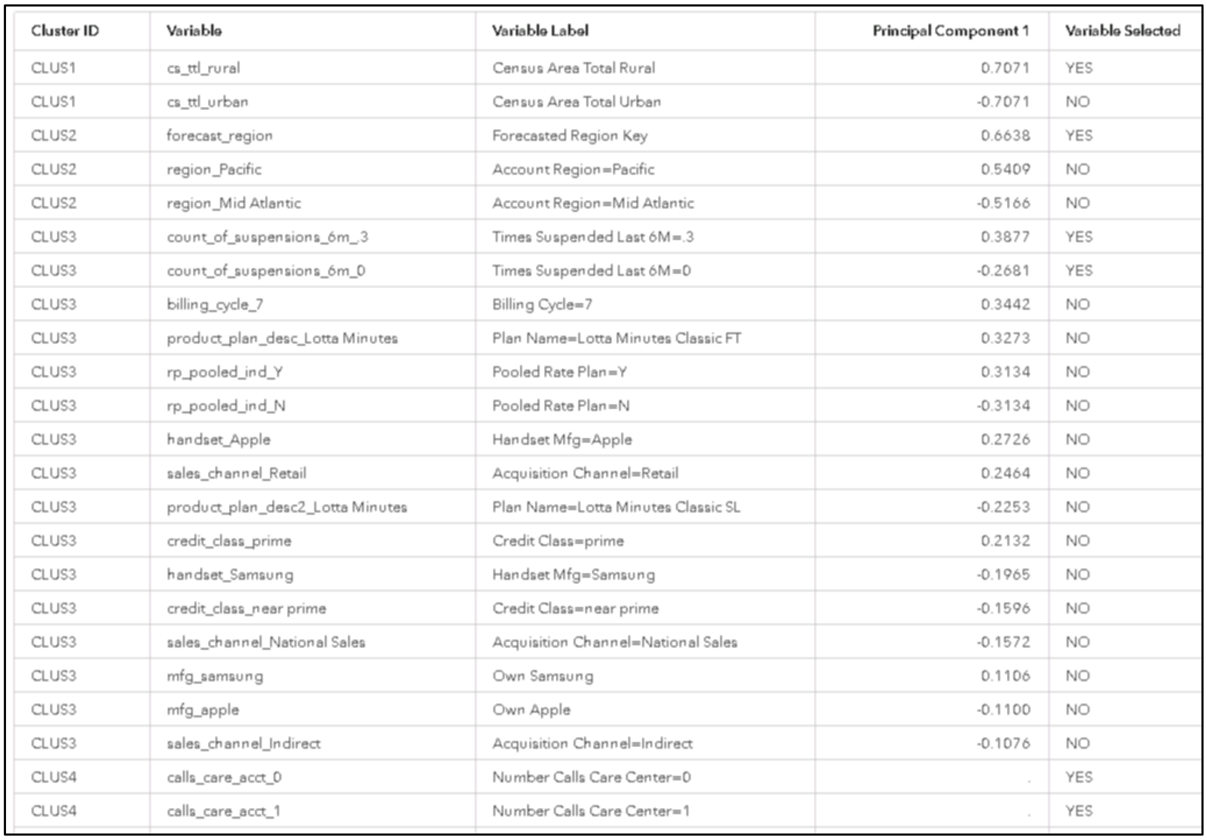
0.8

Note: You use Rho to control the sparsity of connections among variables. Tuning the regularization parameter from low to high increases the number of disconnected components and splits larger clusters into smaller ones. Those divided clusters naturally form a hierarchical structure during this process.

5. Run the Variable Clustering node and view the results when it is complete.

Expand the **Clustered Variables** table. This table contains all the clustered variables, a list of their cluster IDs, variable labels, first principal components, and whether they were selected. Only a subset of the selected variables is shown below.

Note: For class variables, the principal component might be blank. This is valid and expected.



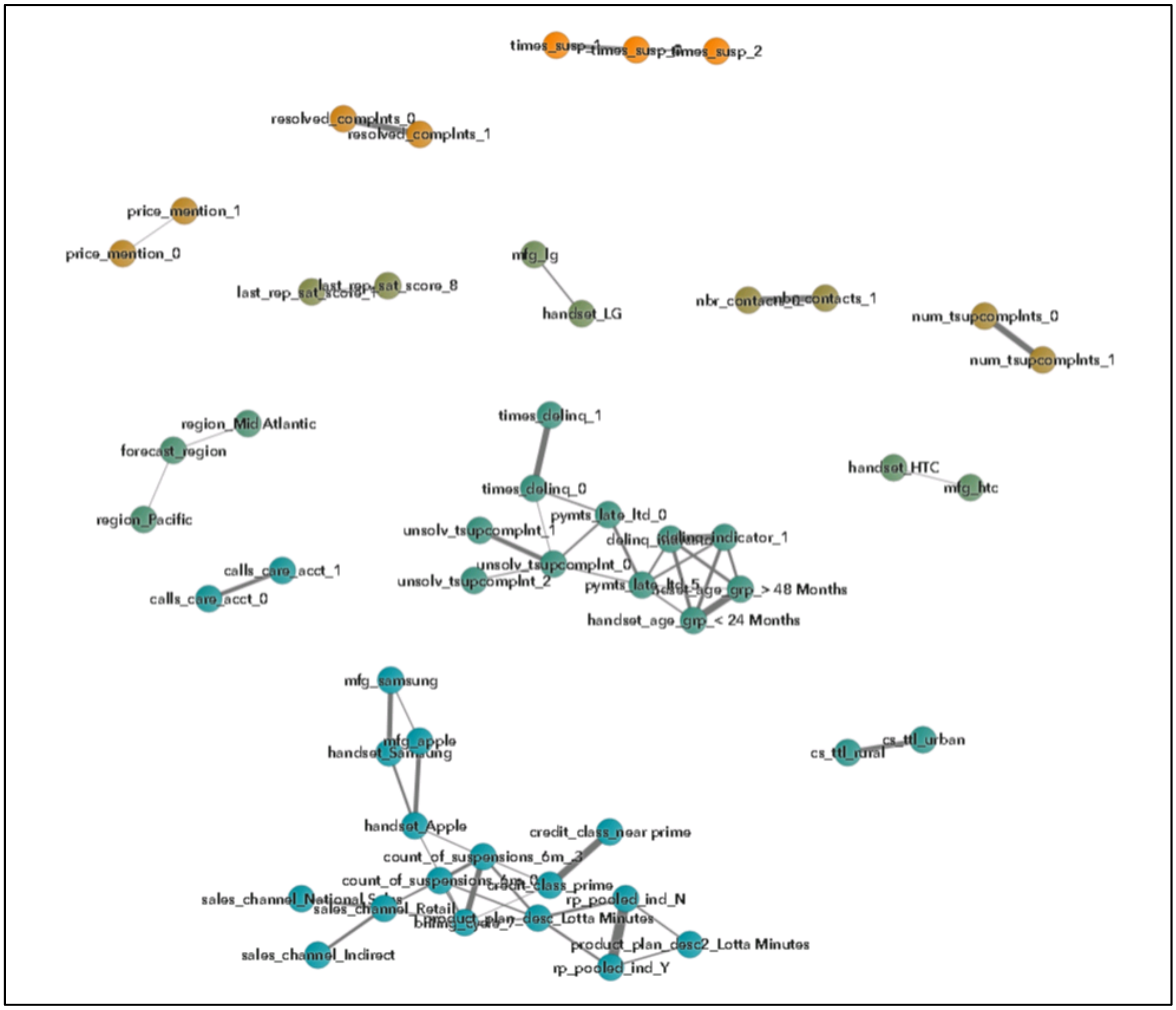
Cluster ID	Variable	Variable Label	Principal Component 1	Variable Selected
CLUS1	cs_ttl_rural	Census Area Total Rural	0.7071	YES
CLUS1	cs_ttl_urban	Census Area Total Urban	-0.7071	NO
CLUS2	forecast_region	Forecasted Region Key	0.6638	YES
CLUS2	region_Pacific	Account Region=Pacific	0.5409	NO
CLUS2	region_Mid Atlantic	Account Region=Mid Atlantic	-0.5166	NO
CLUS3	count_of_suspensions_6m_3	Times Suspended Last 6M=.3	0.3877	YES
CLUS3	count_of_suspensions_6m_0	Times Suspended Last 6M=0	-0.2681	YES
CLUS3	billing_cycle_7	Billing Cycle=7	0.3442	NO
CLUS3	product_plan_desc_Lotta Minutes	Plan Name=Lotta Minutes Classic FT	0.3273	NO
CLUS3	rp_pooled_ind_Y	Pooled Rate Plan=Y	0.3134	NO
CLUS3	rp_pooled_ind_N	Pooled Rate Plan=N	-0.3134	NO
CLUS3	handset_Apple	Handset Mfg=Apple	0.2726	NO
CLUS3	sales_channel_Retail	Acquisition Channel=Retail	0.2464	NO
CLUS3	product_plan_desc2_Lotta Minutes	Plan Name=Lotta Minutes Classic SL	-0.2253	NO
CLUS3	credit_class_prime	Credit Class=prime	0.2132	NO
CLUS3	handset_Samsung	Handset Mfg=Samsung	-0.1965	NO
CLUS3	credit_class_near prime	Credit Class=near prime	-0.1596	NO
CLUS3	sales_channel_National Sales	Acquisition Channel=National Sales	-0.1572	NO
CLUS3	mfg_samsung	Own Samsung	0.1106	NO
CLUS3	mfg_apple	Own Apple	-0.1100	NO
CLUS3	sales_channel_Indirect	Acquisition Channel=Indirect	-0.1076	NO
CLUS4	calls_care_acct_0	Number Calls Care Center=0	.	YES
CLUS4	calls_care_acct_1	Number Calls Care Center=1	.	YES

The first column has cluster membership against each variable. Scrolling down in the Clustered Variables table shows that there are nearly 13 clusters created. Each cluster has different number of inputs. The last column has information about the input variables selected (YES) or not selected (NO) within each cluster by the node. You can verify that the original class variables are kept or dropped in the selection process and not the dummy variables.

Graphical LASSO based on Friedman, Hastie, and Tibshirani (2008) is performed. It estimates the inverse covariance matrix at a specified regularization parameter (ρ=0.8 in this case). The inverse covariance matrix interprets the partial correlation between variables given other variables. Conditional dependency among variables is interpreted by estimating the inverse covariance matrix. The off-diagonal elements of an inverse covariance matrix correspond to partial correlations, so the zero elements imply conditional independence between the pair of variables. The conditional independence provides a better model for understanding the direct link between variables than does simple correlation analysis, which models each pair of variables without considering other variables.

6. Restore the view of the Clustered Variables table.

7. Expand the **Clustered Variables** network.



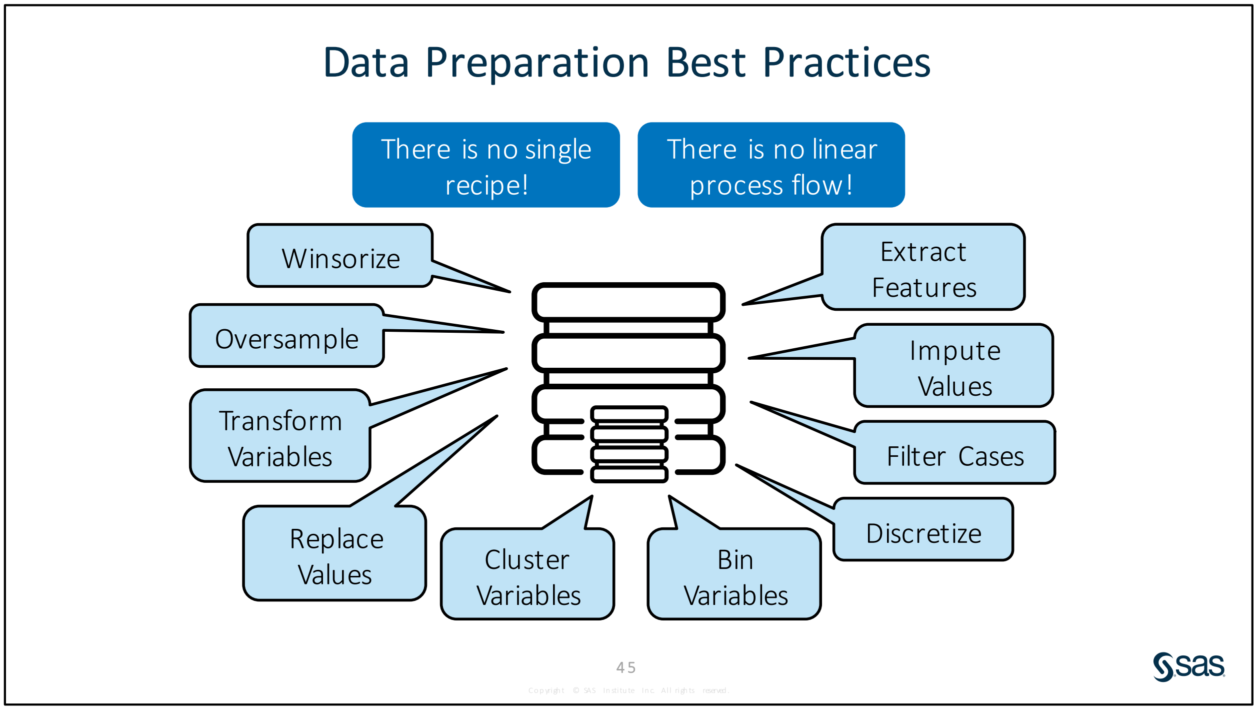
This is a spatial map that gives the orientation and relative distance of clusters and cluster members. Cluster members with a stronger link are connected by a thicker line. There are clearly ***13*** clusters standing out in different color shades. Many of them seem to be closer and consequently similar.

8. Restore the view of the Clustered Variables Network.

9. Close the Results.

End of Demonstration

2.6 Best Practices



Data Preparation Best Practices

There is no single recipe!

There is no linear process flow!

45

Data preprocessing covers a range of processes that are different for raw, structured, and unstructured data (from one or multiple sources). Data preprocessing processes focus on improving the quality of data and its completeness, standardizing how it is defined and structured, collecting and consolidating it, and taking transformation steps to make it useful, particularly for machine learning analysis. The selection and type of preparation processes can differ depending on your purpose, your data expertise, how you plan to interact with the data, and what type of questions you want to answer.

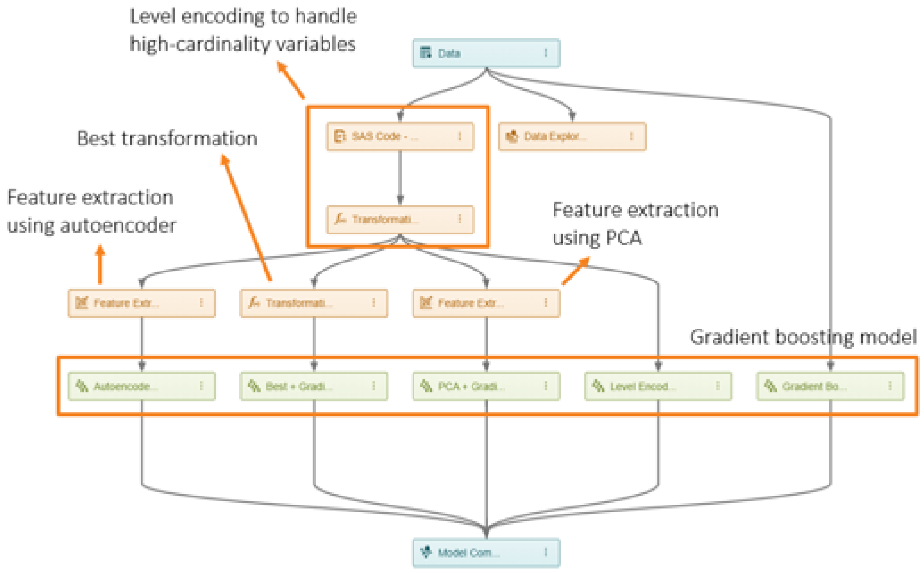
The table below summarizes some challenges that you might encounter in preparing your data. It also includes suggestions for how to handle the challenge by using the Data Mining Preprocessing pipeline nodes in Model Studio.

Data Problem	Common Challenges	Suggested Best Practice
Data Collection	• Biased Data • Incomplete data • High-dimensional data • Sparsity	• Take time to understand the business problem and its context • Enrich the data • Dimension reduction (Feature Extraction, Variable Clustering, and Variable Selection nodes) • Change representation of data (Transformations node)

Data Problem	Common Challenges	Suggested Best Practice
“Untidy” Data	• Value ranges as columns • Multiple variables in the same column • Variables in both rows and columns	• Transform the data with SAS code (Code node)
Outliers	• Out-of-range numeric values and unknown categorical values in score data	• Discretization (Transformations node) • Winsorizing (Imputation node)
Sparse target variables	• Low primary event occurrence rate • Overwhelming preponderance of zero or missing values in target	• Proportional oversampling
Variables of disparate magnitudes	• Misleading variable importance • Distance measure imbalance • Gradient dominance	• Standardization (Transformations node)
High-cardinality variables	• Overfitting • Unknown categorical values in holdout data	• Binning (Transformations node) • Replacement (Replacement node)
Missing Data	• Information loss • Bias	• Binning (Transformations node) • Imputation (Imputation node)
Strong multicollinearity	• Unstable parameter estimates	• Dimension reduction (Feature Extraction, Variable Clustering, and Variable Selection nodes)

Note: Some of these challenges can also be handled in the modeling stage, such as using tree-based methods for handling missing data automatically, which is discussed in subsequent chapters.

Automated Feature Engineering Pipeline Template



Level encoding to handle high-cardinality variables

Best transformation

Feature extraction using autoencoder

Feature extraction using PCA

Gradient boosting model

Shown above is the automated feature engineering pipeline template in Model Studio. Whether you perform feature selection or feature extraction, your goal is to include the subset of features that describe most, but not all, of the variance and to reduce the signal-to-noise ratio in your data. Although intuition would tell you that elimination of features equates to a loss of information, in the end this loss is compensated for by the ability of the model to more accurately map the remaining features to the target in a lower-dimensional space. The result is simpler models, shorter training times, improved generalization, and a greater ability to visualize the feature space.

Some high-dimensional data sets require special attention to perform feature extraction efficiently. One example is a data set of user ratings for items (such as movies) in which each column represents an item and each row is a user (or vice versa). In the template above, the SAS code node makes sure that the variables with high cardinality are not rejected from the analysis (by default Model Studio rejects nominal variables that has more than 20 levels) by increasing the highest acceptable cardinality level to 1000. This node also specifies level encoding transformation in the metadata for these high-cardinality variables (nominals that have cardinality between 20 and 1000). The Transformations node connected to the SAS code node performs a level encoding transformation that transforms these high cardinality nominal variables to numeric. This is a simple transformation that assigns numeric values for each level of the nominal variable according to an alphabetical order. Level encoding is not an ideal transformation, but it works well in terms of improving model accuracy when compared to excluding those variables from the analysis. Note that transformations specified in the metadata would take place only if you run a Transformation node after this specification. For this reason, in the Transformations node that is connected to the SAS Code node, there is no transformation specified for Class Inputs. Instead, it is set to **None**. Here the Transformation code's role is simply to implement the transformation (or transformations) specified in the metadata.

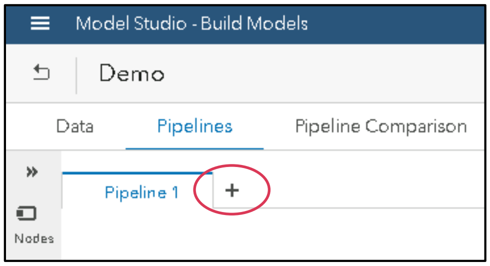
More effective transformations for high cardinality variables include target-based transformations such as creating a feature that captures the frequency of the occurrence of each level of the nominal variable. For high cardinality, this helps a lot! You might use ratio or percentage of a level to all the levels present. Similarly, you can encode a high-cardinality variable by using another numeric input variable by choosing the max, min or median value of that variable for each level of the high-cardinality nominal variable.

Note: For more details, see “Automate your feature engineering.” (https://blogs.sas.com/content/subconsciousmusings/2018/08/09/automate-your-feature-engineering/)

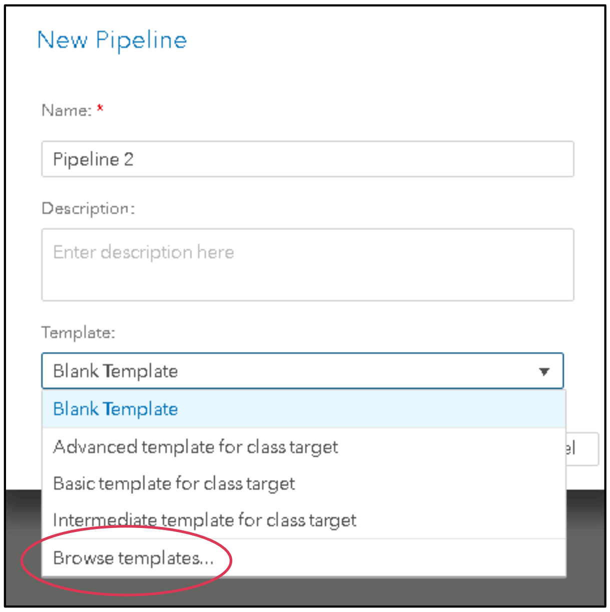
Running the Automated Feature Engineering Pipeline Template (Self-Study)

In this demonstration, you run the automated feature engineering pipeline template on **commsdata**.

1. Click + next to the current pipeline tab in the upper left corner of the canvas.



2. In the New Pipeline window, under Template, select **Browse templates**.


New Pipeline

Name: * Pipeline 2

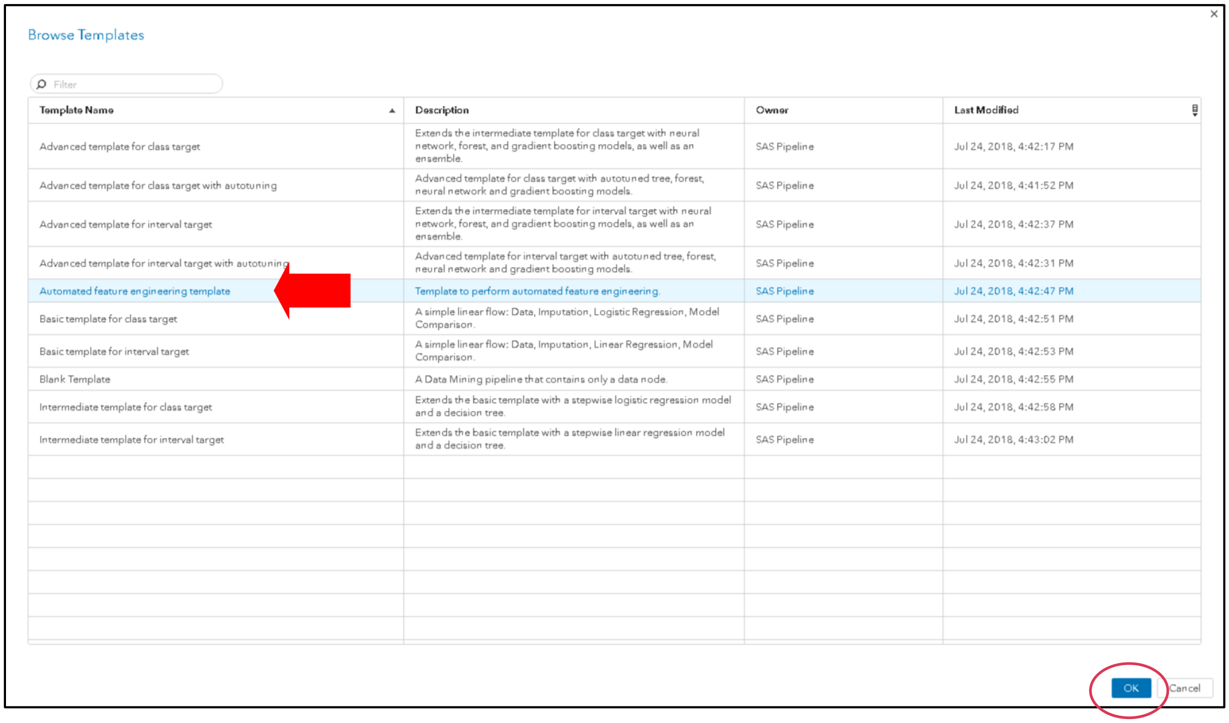
Description: Enter description here

Template: Blank Template

Blank Template
Advanced template for class target
Basic template for class target
Intermediate template for class target
Browse templates...

Note: Some of the options on the Templates menu might be different on your classroom computer.

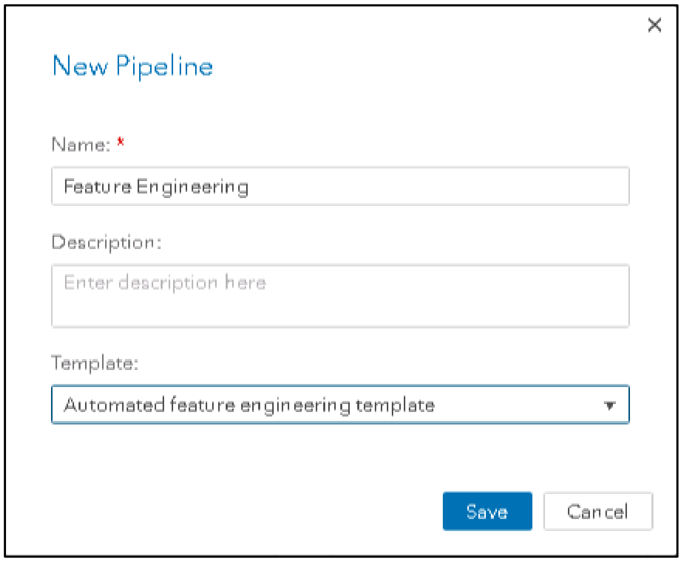
3. In the Browse Templates window, select **Automated feature engineering template**. Click **OK**.



Browse Templates

Template Name	Description	Owner	Last Modified
Advanced template for class target	Extends the intermediate template for class target with neural network, forest, and gradient boosting models, as well as an ensemble.	SAS Pipeline	Jul 24, 2018, 4:42:17 PM
Advanced template for class target with autotuning	Advanced template for class target with autotuned tree, forest, neural network and gradient boosting models.	SAS Pipeline	Jul 24, 2018, 4:41:52 PM
Advanced template for interval target	Extends the intermediate template for interval target with neural network, forest, and gradient boosting models, as well as an ensemble.	SAS Pipeline	Jul 24, 2018, 4:42:37 PM
Advanced template for interval target with autotuning	Advanced template for interval target with autotuned tree, forest, neural network and gradient boosting models.	SAS Pipeline	Jul 24, 2018, 4:42:31 PM
Automated feature engineering template	Template to perform automated feature engineering.	SAS Pipeline	Jul 24, 2018, 4:42:47 PM
Basic template for class target	A simple linear flow: Data, Imputation, Logistic Regression, Model Comparison.	SAS Pipeline	Jul 24, 2018, 4:42:51 PM
Basic template for interval target	A simple linear flow: Data, Imputation, Linear Regression, Model Comparison.	SAS Pipeline	Jul 24, 2018, 4:42:53 PM
Blank Template	A Data Mining pipeline that contains only a data node.	SAS Pipeline	Jul 24, 2018, 4:42:55 PM
Intermediate template for class target	Extends the basic template with a stepwise logistic regression model and a decision tree.	SAS Pipeline	Jul 24, 2018, 4:42:58 PM
Intermediate template for interval target	Extends the basic template with a stepwise linear regression model and a decision tree.	SAS Pipeline	Jul 24, 2018, 4:43:02 PM

4. In the New Pipeline window, enter the name **Feature Engineering**.



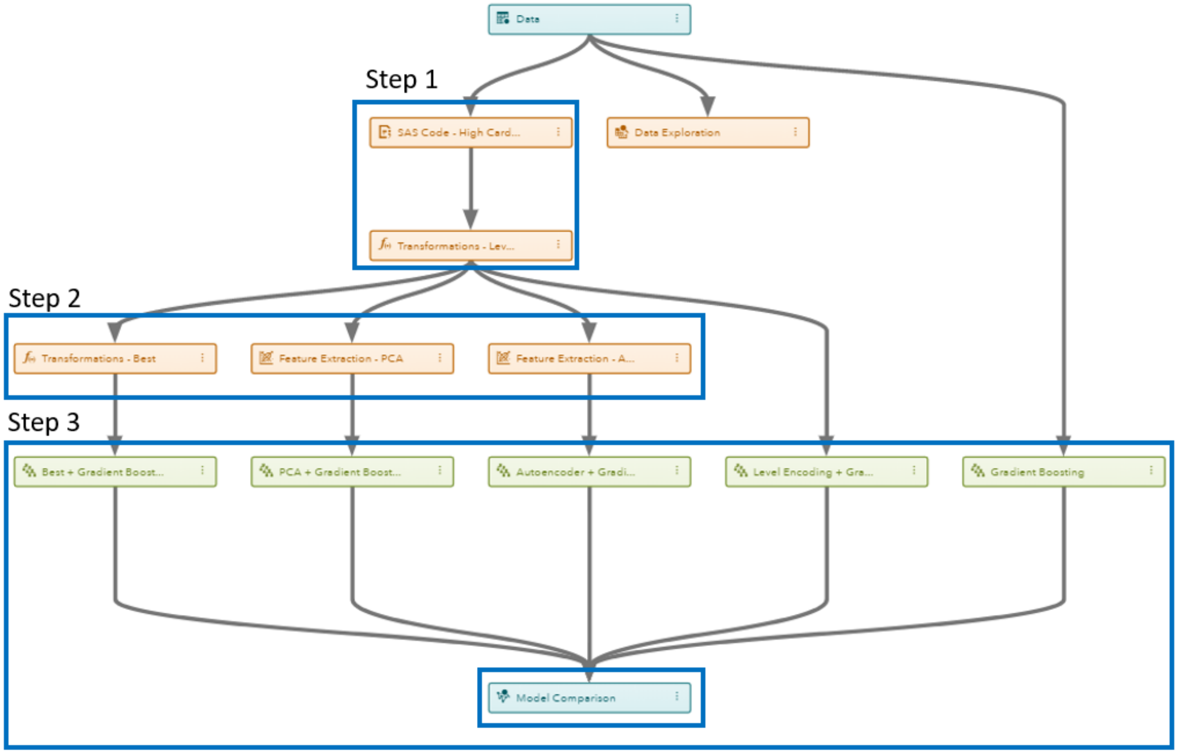
New Pipeline

Name: * Feature Engineering

Description: Enter description here

Template: Automated feature engineering template

5. Click **Save**.



The template automatically creates engineered features by using popular feature transformation and extraction techniques. The idea is to automatically learn a set of features (from potentially noisy, raw data) that can be useful in supervised learning tasks without manually creating engineered features. These are the steps of the template:

Step 1: Perform level encoding for high-cardinality variables.

Step 2: Create new features by using the Best transformation, PCA / SVD, and autoencoder methods.

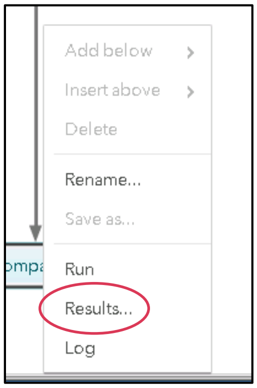
Step 3: Compare the predictive performance of the newly engineered features to original features. The five feature sets are used as inputs for the gradient boosting algorithm. The gradient boosting algorithm is used because it is an effective supervised learning algorithm that often outperforms other algorithms in terms of predictive accuracy.

Note: A more detailed explanation of the template can be found in “3 steps of the automated feature engineering template in SAS.” (https://communities.sas.com/t5/SAS-Communities-Library/3-steps-of-the-automated-feature-engineering-template-in-SAS/ta-p/484383)

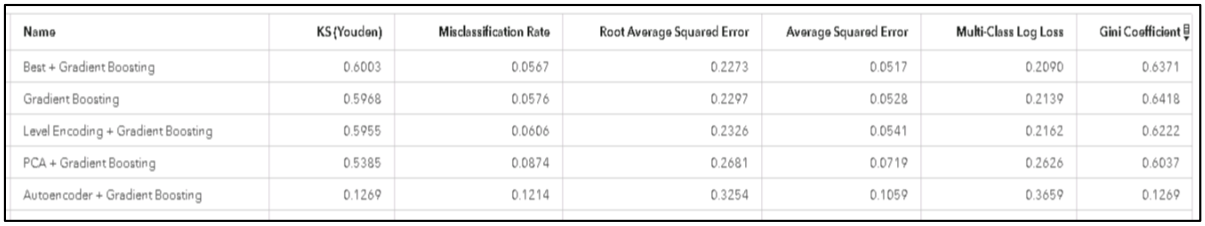
6. Select each node one-by-one, right-click, and select **Run**. Do ***not*** click the Run pipeline icon.

   Note: Running this template can substantially increase run time. Remember that limited resources are available in your class environment. Automatic hyperparameter tuning (autotuning) is turned on to find the optimal hyperparameter settings of the gradient boosting algorithm, so the comparison between feature sets is fairer and not dependent on the hyperparameters. However, keep in mind that autotuning comes with an additional computing cost. If this step takes too long to run, you can change the autotuning settings, or simply turn it off and use the default hyperparameter settings. With the default settings in the template, we recommend running this pipeline node-by-node. You might encounter an error of insufficient resources if you run the entire pipeline.

7. After the pipeline has successfully run, right-click the **Model Comparison** node and select **Results**.



8. Click ⤢ to expand the Model Comparison table. Unless specified, the default fit statistic (KS) is used for selecting a champion model with a class target.



Name	KS (Youden)	Misclassification Rate	Root Average Squared Error	Average Squared Error	Multi-Class Log Loss	Gini Coefficient
Best + Gradient Boosting	0.6003	0.0567	0.2273	0.0517	0.2090	0.6371
Gradient Boosting	0.5968	0.0576	0.2297	0.0528	0.2139	0.6418
Level Encoding + Gradient Boosting	0.5955	0.0606	0.2326	0.0541	0.2162	0.6222
PCA + Gradient Boosting	0.5385	0.0874	0.2681	0.0719	0.2626	0.6037
Autoencoder + Gradient Boosting	0.1269	0.1214	0.3254	0.1059	0.3659	0.1269

   Explore the results.

   You compare the performance of the five different feature sets (three automatically engineered sets, the original set with level encoding for the high-cardinality variables, and the original set without level encoding).

   It is important to remember that using this template does not guarantee that one of the automatically created feature sets performs better than the original features for your data, because every data set is unique and this template uses only a few techniques. Instead, the goal of this is to show an example of how you can create different automatically engineered feature sets by using many other tools provided in Model Studio and test their performance in a similar way with a minimal effort.

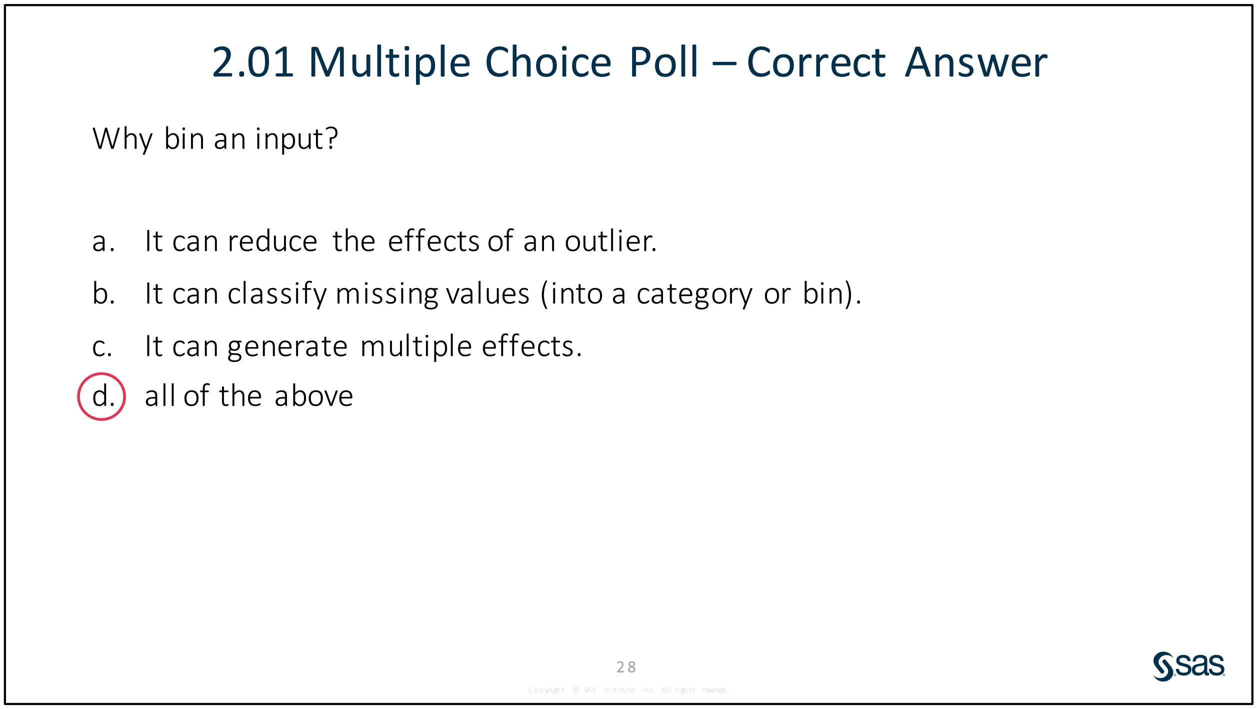
9. Click ⤡ to exit the maximized view.

10. Click **Close** to close the Model Comparison Results window.

End of Demonstration

2.7 Solutions

Solutions to Student Activities (Polls/Quizzes)



2.01 Multiple Choice Poll – Correct Answer

Why bin an input?

a. It can reduce the effects of an outlier.

b. It can classify missing values (into a category or bin).

c. It can generate multiple effects.

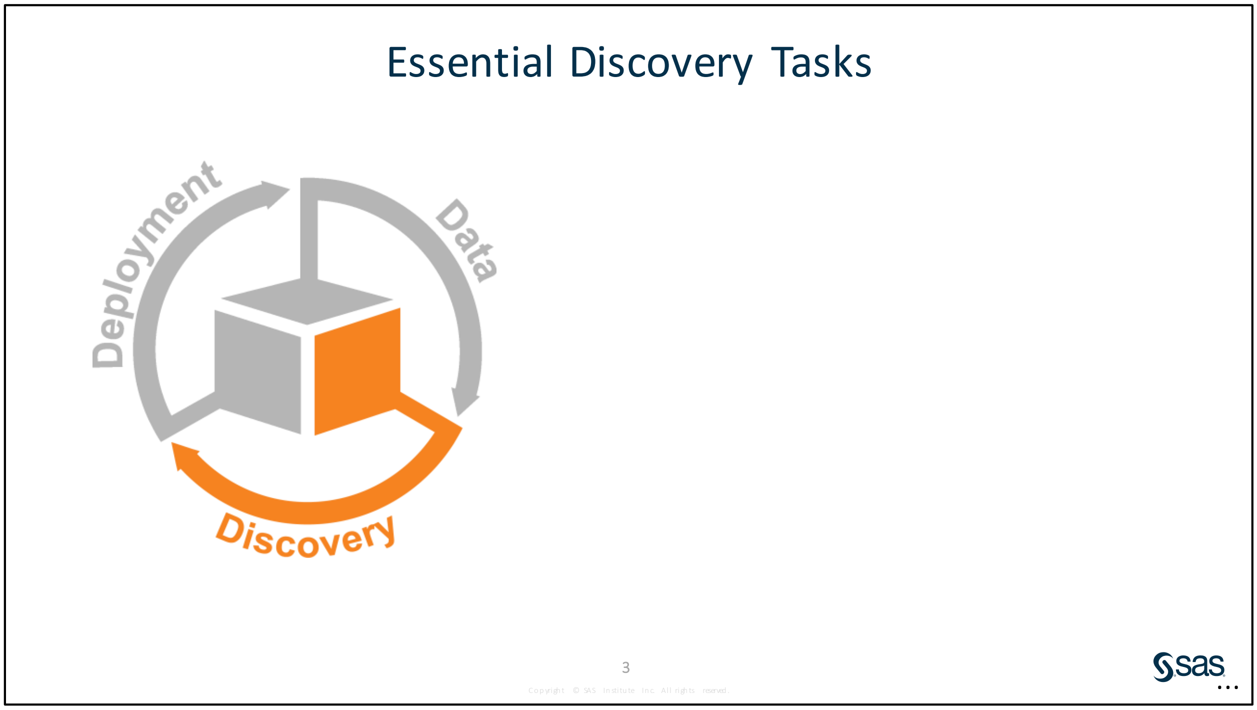
[✓ d.] all of the above

28

Copyright © SAS Institute Inc. All rights reserved.

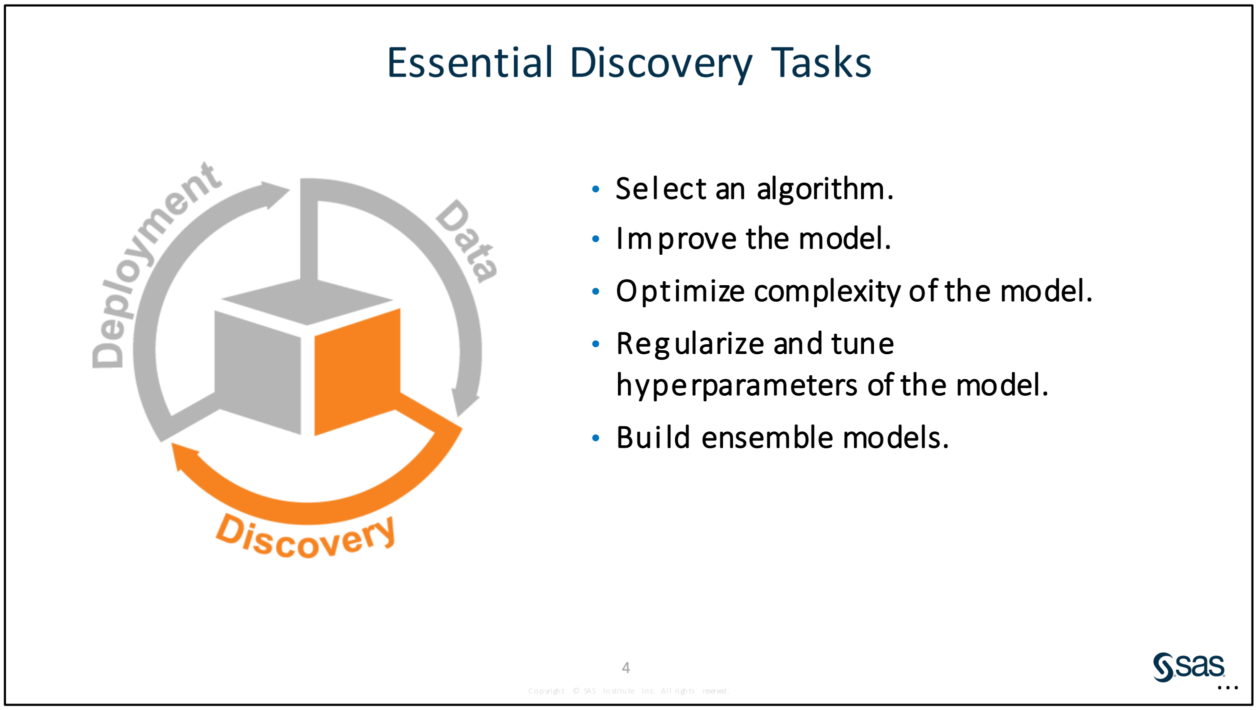
Chapter 3 Decision Trees and Ensembles of Trees

3.1 Introduction



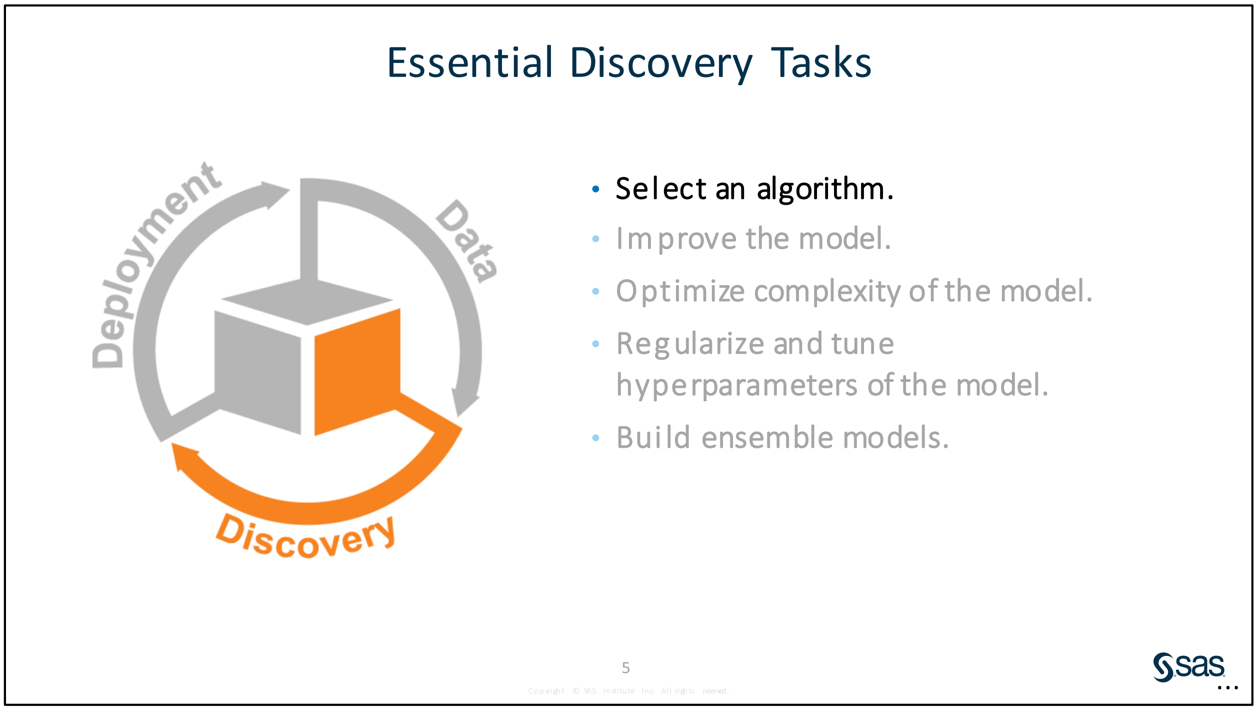
Essential Discovery Tasks

Now that you have ensured that you have sufficient and appropriate data, massaged the data into a form suitable for modeling, identified key features to include in your model, and established how the model is to be used, you are ready to use powerful machine learning algorithms to build predictive models or discover patterns in your data. This is really the phase where you should allow yourself more freedom to experiment with different approaches to identify the algorithms (and configuration of options for those algorithms) that produce the best model for your specific application.



Essential Discovery Tasks

- Select an algorithm.
- Improve the model.
- Optimize complexity of the model.
- Regularize and tune hyperparameters of the model.
- Build ensemble models.



Essential Discovery Tasks

- Select an algorithm.
- Improve the model.
- Optimize complexity of the model.
- Regularize and tune hyperparameters of the model.
- Build ensemble models.

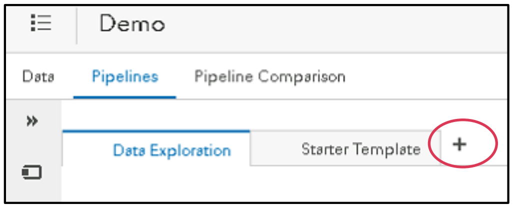
The success of your machine learning application comes down to the effectiveness of the actual model that you build. The popular “no free lunch” theorem (Wolpert 1996) states that no one model works best for every problem. We will start building predictive model by training a decision tree.

Note: Selecting your algorithm is discussed in a broader sense at the end of Chapter 5 when we finish training several algorithms.

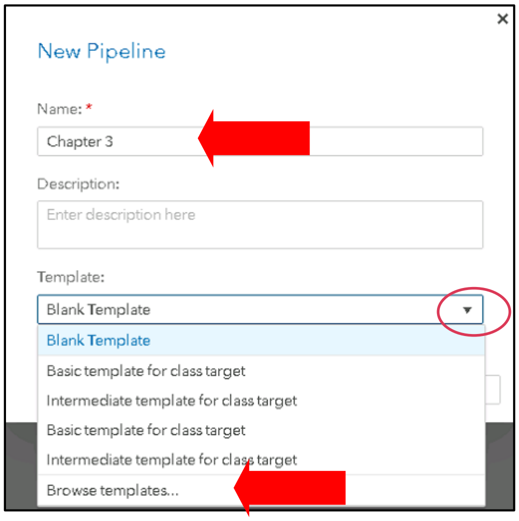
Building a Decision Tree Model with Default Settings

In this demonstration, you use the CPML demo pipeline as a starting place in a new pipeline in the Demo project. You add a Decision Tree node and build a Decision Tree model using the default settings of the node.

1. Return to the Demo project and click the **Pipelines** tab. Click the plus sign (**+**) next to the Starter Template tab to add a new pipeline.



2. In the New Pipeline window, enter **Chapter 3** in the **Name** field. For **Template**, select **Browse templates**.


New Pipeline

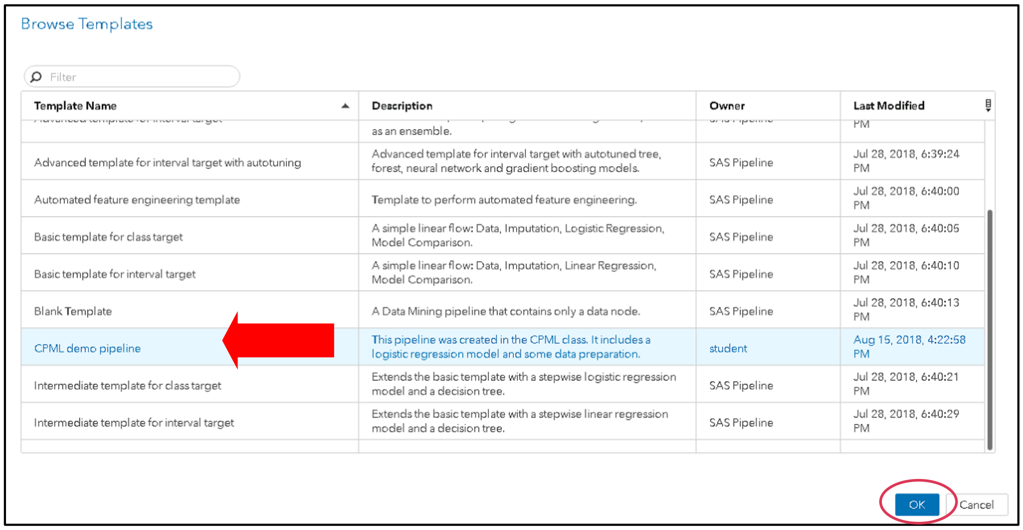
Name: * Chapter 3

Description: Enter description here

Template: Blank Template

Blank Template
Basic template for class target
Intermediate template for class target
Basic template for class target
Intermediate template for class target
Browse templates...

3. Scroll down as needed in the Browse Template window and select **CPML demo pipeline**. Click **OK**.

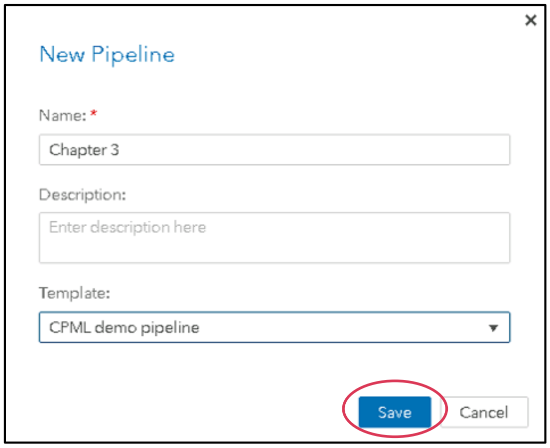


Browse Templates

Filter

Template Name	Description	Owner	Last Modified
	as an ensemble.		PM
Advanced template for interval target with autotuning	Advanced template for interval target with autotuned tree, forest, neural network and gradient boosting models.	SAS Pipeline	Jul 28, 2018, 6:39:24 PM
Automated feature engineering template	Template to perform automated feature engineering.	SAS Pipeline	Jul 28, 2018, 6:40:00 PM
Basic template for class target	A simple linear flow: Data, Imputation, Logistic Regression, Model Comparison.	SAS Pipeline	Jul 28, 2018, 6:40:05 PM
Basic template for interval target	A simple linear flow: Data, Imputation, Linear Regression, Model Comparison.	SAS Pipeline	Jul 28, 2018, 6:40:10 PM
Blank Template	A Data Mining pipeline that contains only a data node.	SAS Pipeline	Jul 28, 2018, 6:40:13 PM
CPML demo pipeline	This pipeline was created in the CPML class. It includes a logistic regression model and some data preparation.	student	Aug 15, 2018, 4:22:58 PM
Intermediate template for class target	Extends the basic template with a stepwise logistic regression model and a decision tree.	SAS Pipeline	Jul 28, 2018, 6:40:21 PM
Intermediate template for interval target	Extends the basic template with a stepwise linear regression model and a decision tree.	SAS Pipeline	Jul 28, 2018, 6:40:29 PM

4. Click **Save** in the New Pipeline window.



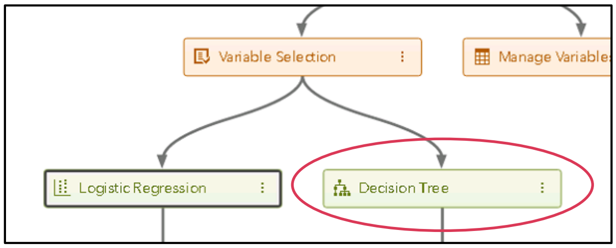
New Pipeline

Name: * Chapter 3

Description: Enter description here

Template: CPML demo pipeline

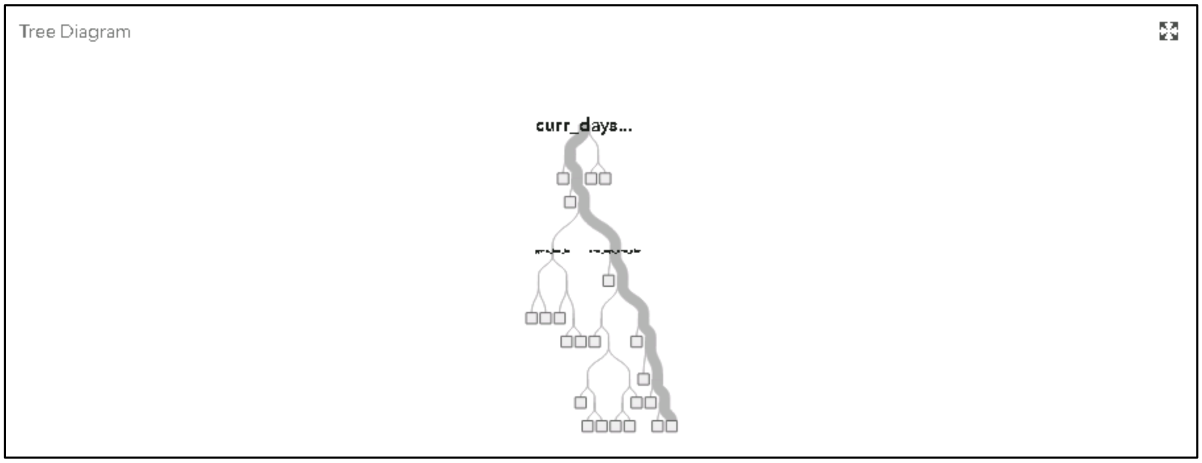
5. In the Chapter 3 pipeline, right-click the **Variable Selection** node and select **Add below ⇨ Supervised Learning ⇨ Decision Tree**.



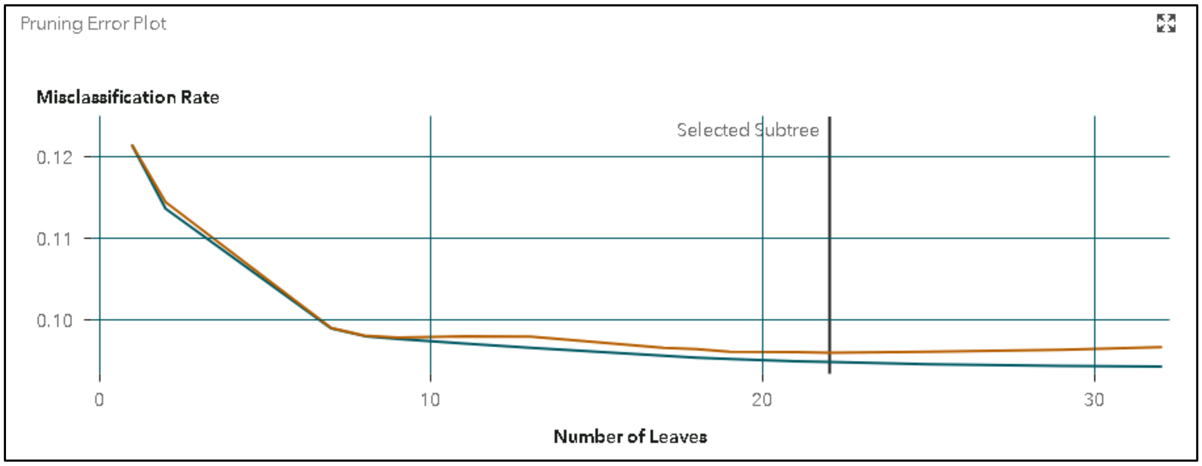
6. Keep all properties for the Decision Tree at their defaults. Run the Decision Tree node.

7. Open the results for the Decision Tree node.

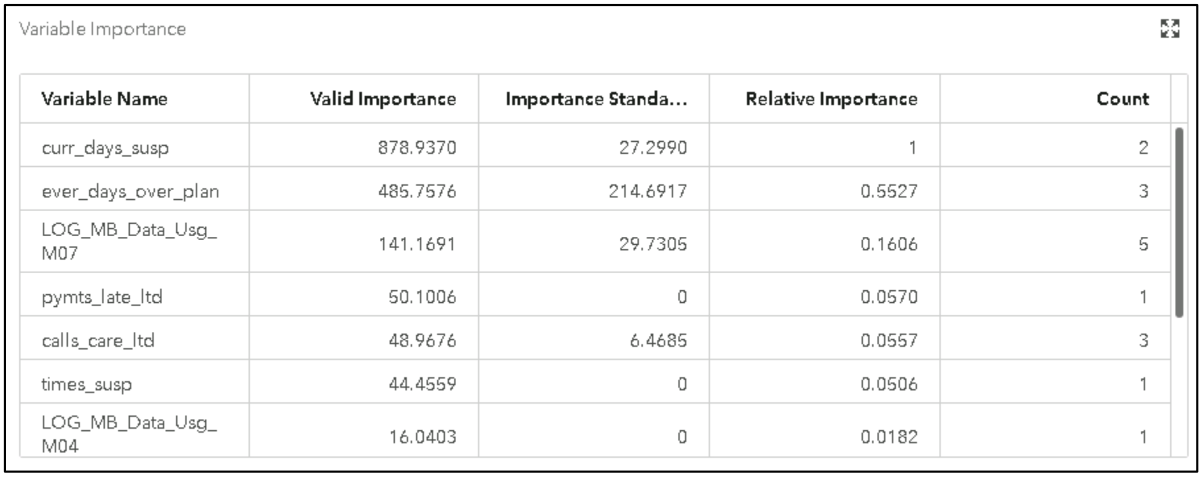
   There are several charts and plots to help you in evaluating the model's performance. The first plot is the Tree Diagram, which presents the final tree structure for this particular model, such as the depth of the tree and all end leaves.



   The Pruning Error Plot shows the model's performance based on the misclassification rate – because the target is binary – throughout the recursive splitting process when new end leaves have been added to the final model.



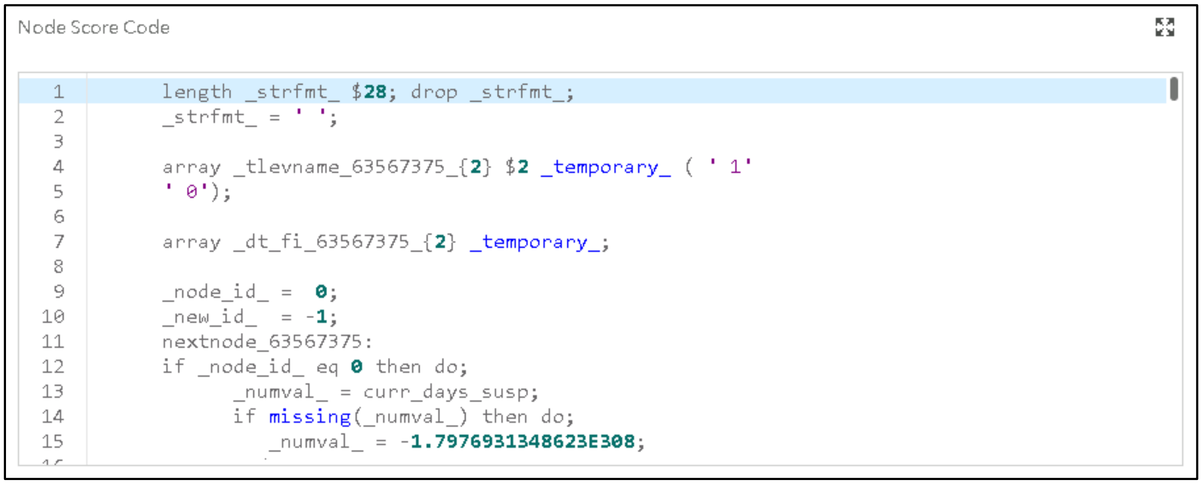
The Variable Importance table shows the input variables most significant to the final model. The most important input variable has its relative importance as one and all others are measured based on the most important input.



Variable Importance

Variable Name	Valid Importance	Importance Standa...	Relative Importance	Count
curr_days_susp	878.9370	27.2990	1	2
ever_days_over_plan	485.7576	214.6917	0.5527	3
LOG_MB_Data_Usg_M07	141.1691	29.7305	0.1606	5
pymts_late_ltd	50.1006	0	0.0570	1
calls_care_ltd	48.9676	6.4685	0.0557	3
times_susp	44.4559	0	0.0506	1
LOG_MB_Data_Usg_M04	16.0403	0	0.0182	1

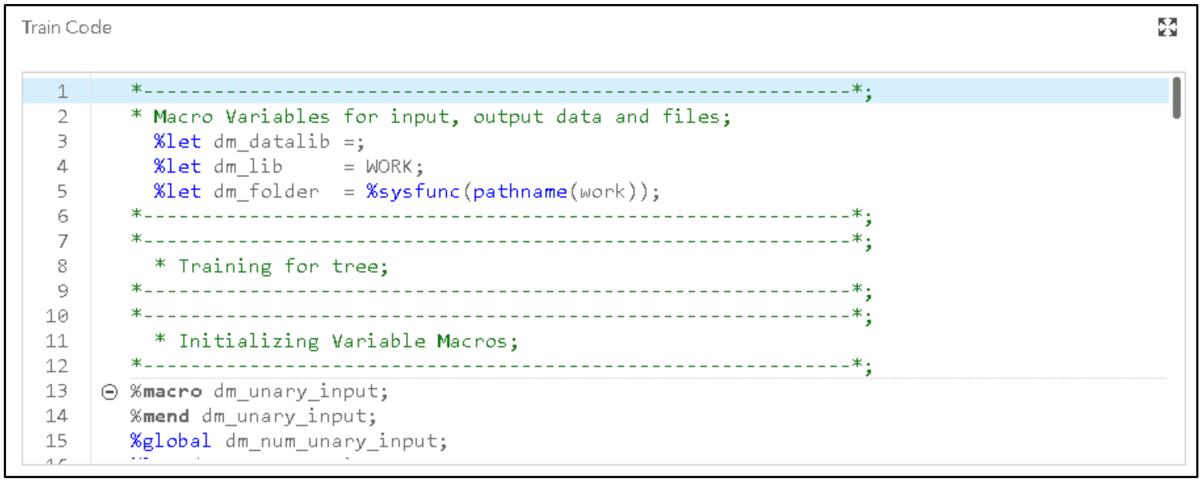
The Node Score Code window shows the final score code that can be deployed in production.



Node Score Code

```
length _strfmt_ $28; drop _strfmt_;
_strfmt_ = ' ';

array _tlevname_63567375_{2} $2 _temporary_ ( ' 1'
' 0');

array _dt_fi_63567375_{2} _temporary_;

_node_id_ =  0;
_new_id_  = -1;
nextnode_63567375:
if _node_id_ eq 0 then do;
      _numval_ = curr_days_susp;
      if missing(_numval_) then do;
         _numval_ = -1.797693134862E308;
```

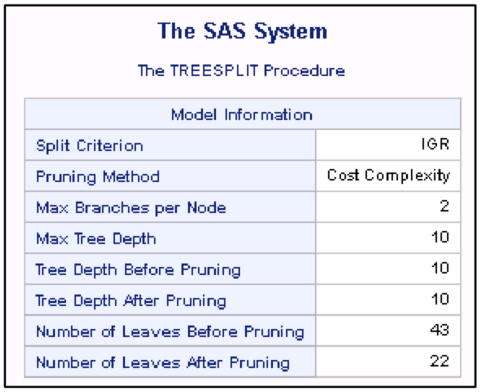
Similarly, the Train Code window shows the train code that can be used to train the model based on different data sets or in different platforms.



Train Code

```
*------------------------------------------------------------*;
* Macro Variables for input, output data and files;
  %let dm_datalib =;
  %let dm_lib     = WORK;
  %let dm_folder  = %sysfunc(pathname(work));
*------------------------------------------------------------*;
*------------------------------------------------------------*;
  * Training for tree;
*------------------------------------------------------------*;
*------------------------------------------------------------*;
  * Initializing Variable Macros;
*------------------------------------------------------------*;
%macro dm_unary_input;
%mend dm_unary_input;
%global dm_num_unary_input;
```

Finally, the Output window shows the final decision tree model parameters, the variable important table, and the pruning iterations.

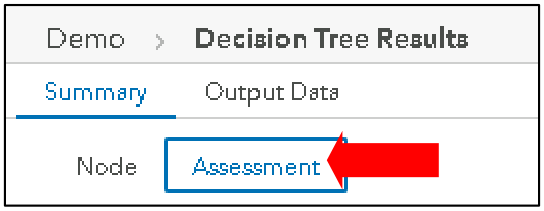


The SAS System

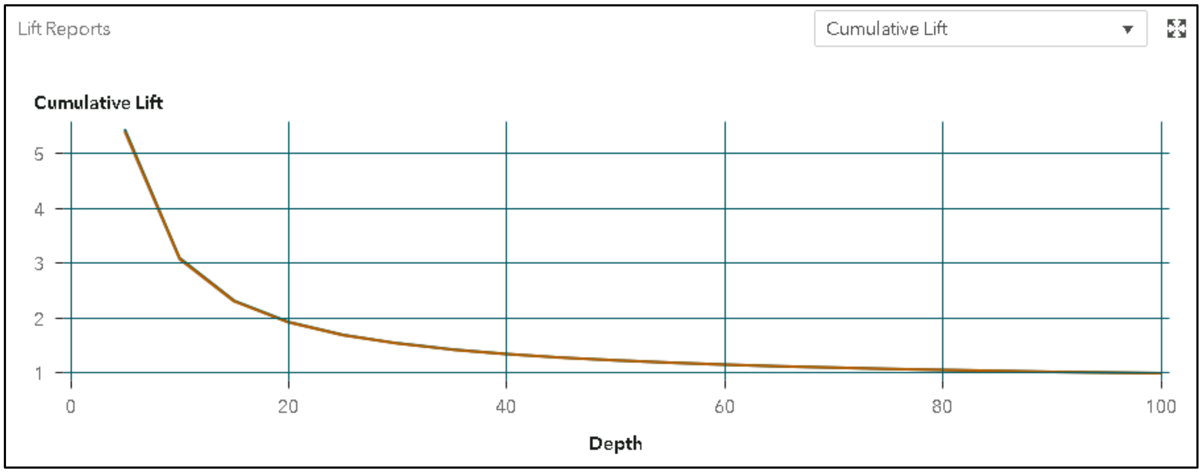
The TREESPLIT Procedure

Model Information	
Split Criterion	IGR
Pruning Method	Cost Complexity
Max Branches per Node	2
Max Tree Depth	10
Tree Depth Before Pruning	10
Tree Depth After Pruning	10
Number of Leaves Before Pruning	43
Number of Leaves After Pruning	22

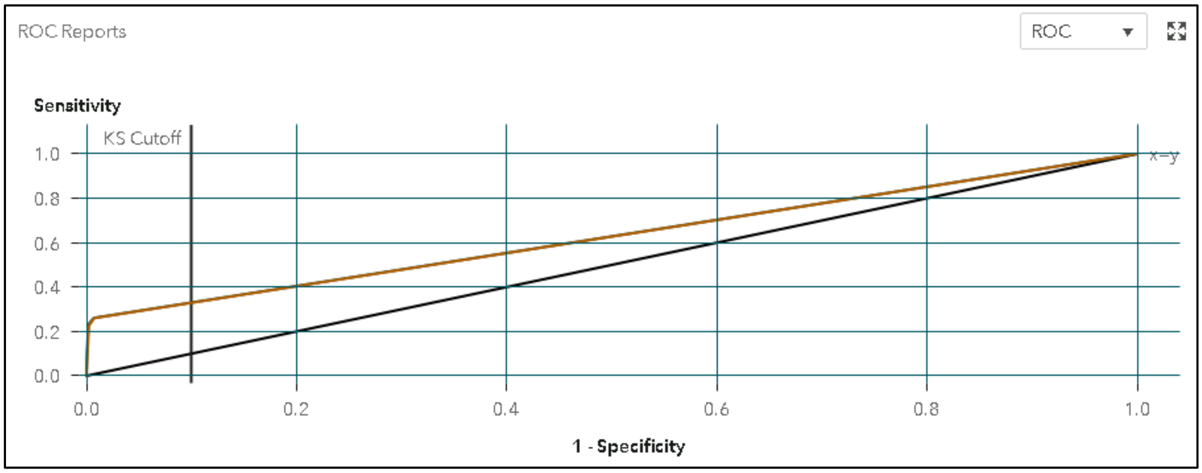
8. Click the **Assessment** tab.



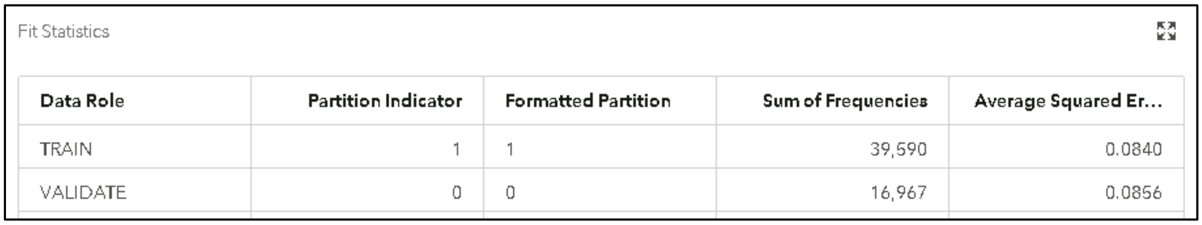
The first chart is the Cumulative Lift, showing the model's performance ordered by the percentage of the population. This chart is very useful for selecting the model based on a particular target of the customer base. It shows how much better the model is than the random events.



For a binary target, you also have the ROC curve, which shows the model's performance considering the true positive rate and the false positive rate. It is good to foresee the performance on a specific business events, when all positive cases are selected. It shows the model's performance based on the positive cases were predicted right and the positive cases were predicted wrong. ROC is very useful for deployment.



Finally, you have the Fit Statistics output, which shows the model's performance based on some assessment measures, such as average square error.



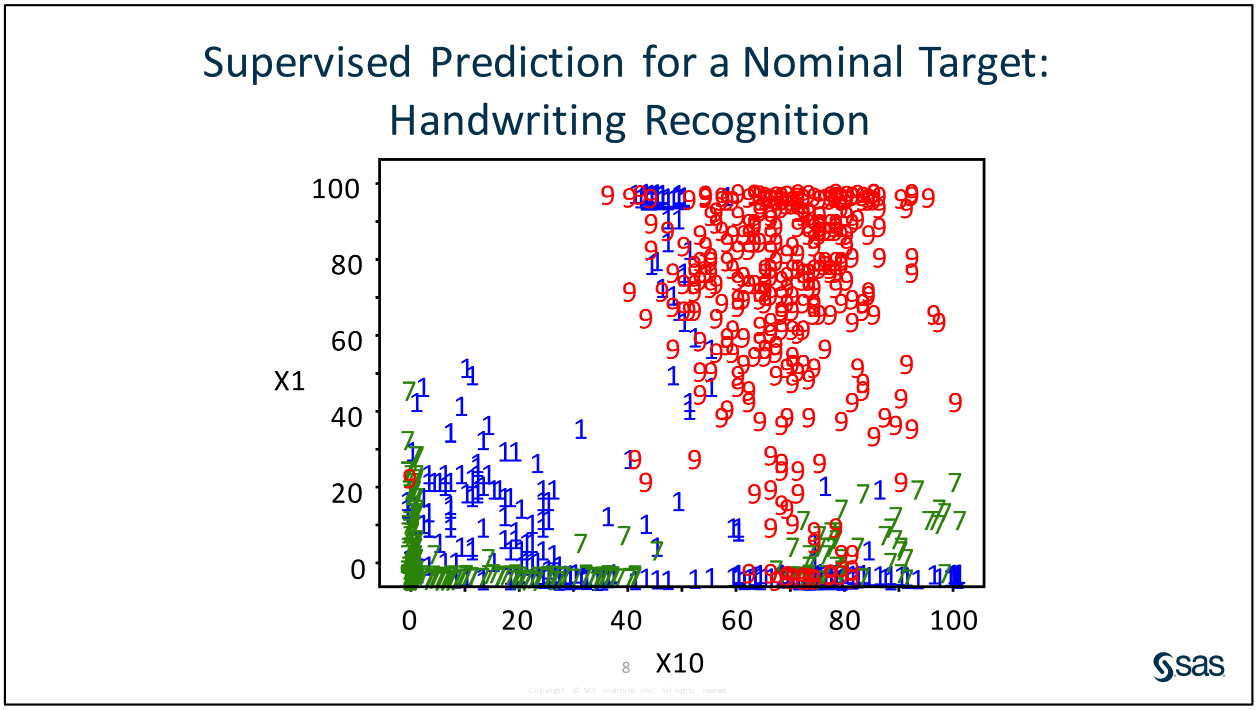
Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.0840
VALIDATE	0	0	16,967	0.0856

The Fit Statistics table shows an average square error of 0.0856 on the VALIDATE partition.

9. Close the Results window.

End of Demonstration

3.2 Tree-Structure Models

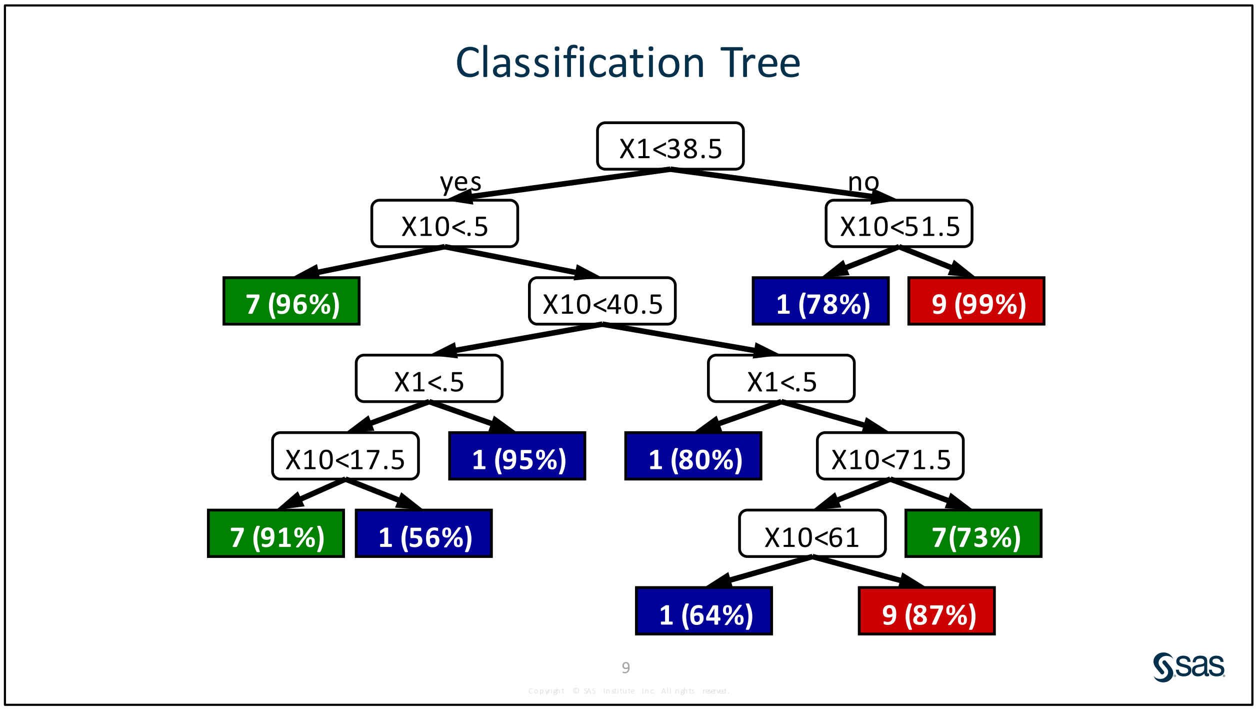


Supervised Prediction for a Nominal Target: Handwriting Recognition

Decision trees are statistical models designed for supervised prediction problems. Supervised prediction encompasses predictive modeling, pattern recognition, discriminant analysis, multivariate function estimation, and supervised machine learning.

Handwriting recognition is a classic application of supervised prediction. The example data set is a subset of the pen-based recognition of handwritten digits data, available from the UCI repository (Blake et al. 1998). The cases are digits written on a pressure-sensitive tablet. The input variables measure the position of the pen. They are scaled to be between 0 and 100. Two of the original sixteen inputs are shown (**X1** and **X10**). The target is the true written digit (0-9). This subset contains the 1064 cases corresponding to the three digits 1, 7, and 9. Each case represents a point in the input space. (The data were jittered for display because many of the points overlap.)

In the pen-digits data, the inputs have an interval measurement scale and the target has a nominal measurement scale. The generic supervised prediction problem places no restrictions on the scales of the inputs or the target.



Classification Tree

X1<38.5

yes

no

X10<.5

X10<51.5

7 (96%)

X10<40.5

1 (78%)

9 (99%)

X1<.5

X1<.5

X10<17.5

1 (95%)

1 (80%)

X10<71.5

7 (91%)

1 (56%)

X10<61

7(73%)

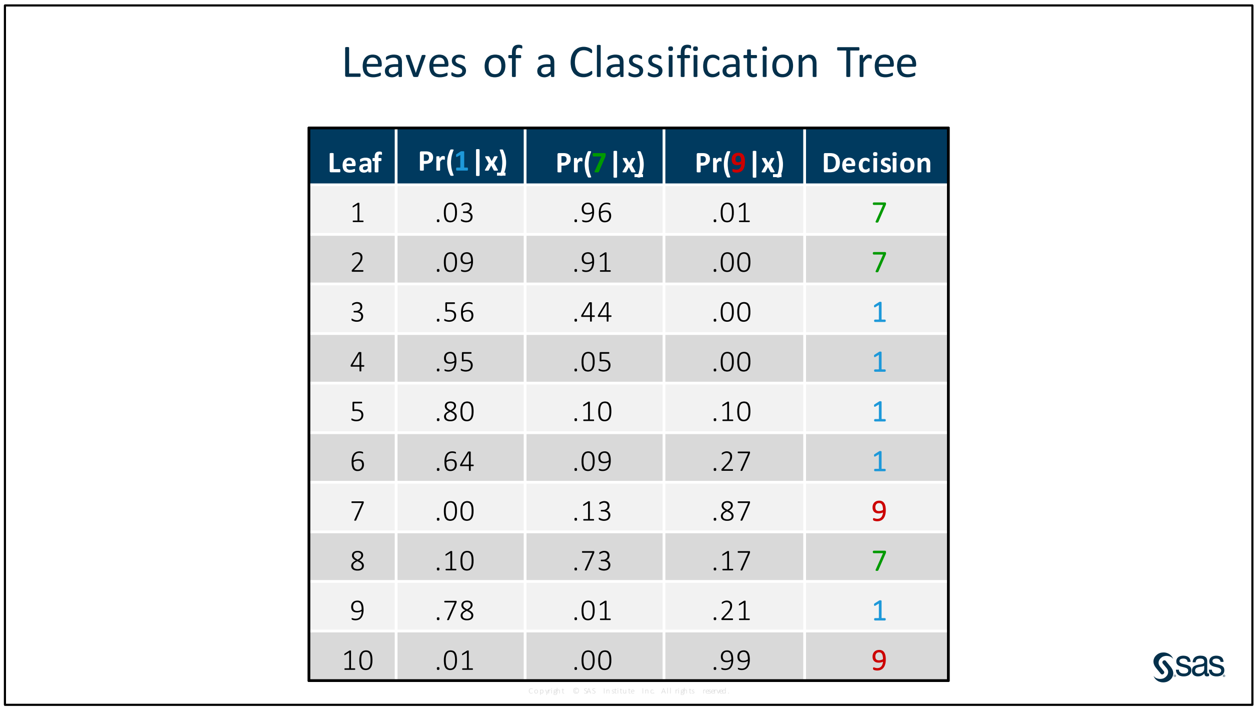
1 (64%)

9 (87%)

9

A decision tree is so called because the predictive model can be represented in a tree-like structure. A decision tree is read from the top down starting at the ***root node***. Each internal node represents a split based on the values of one of the inputs. The inputs can appear in any number of splits throughout the tree. Cases move down the branch that contains its input value. In a binary tree with interval inputs, each internal node is a simple inequality. A case moves left if the inequality is true and right otherwise. The terminal nodes of the tree are called *leaves*. The leaves represent the predicted target. All cases reaching a particular leaf are given the same predicted value. When the target is categorical, the model is a called a *classification tree*. The leaves give the predicted class as well as the probability of class membership.

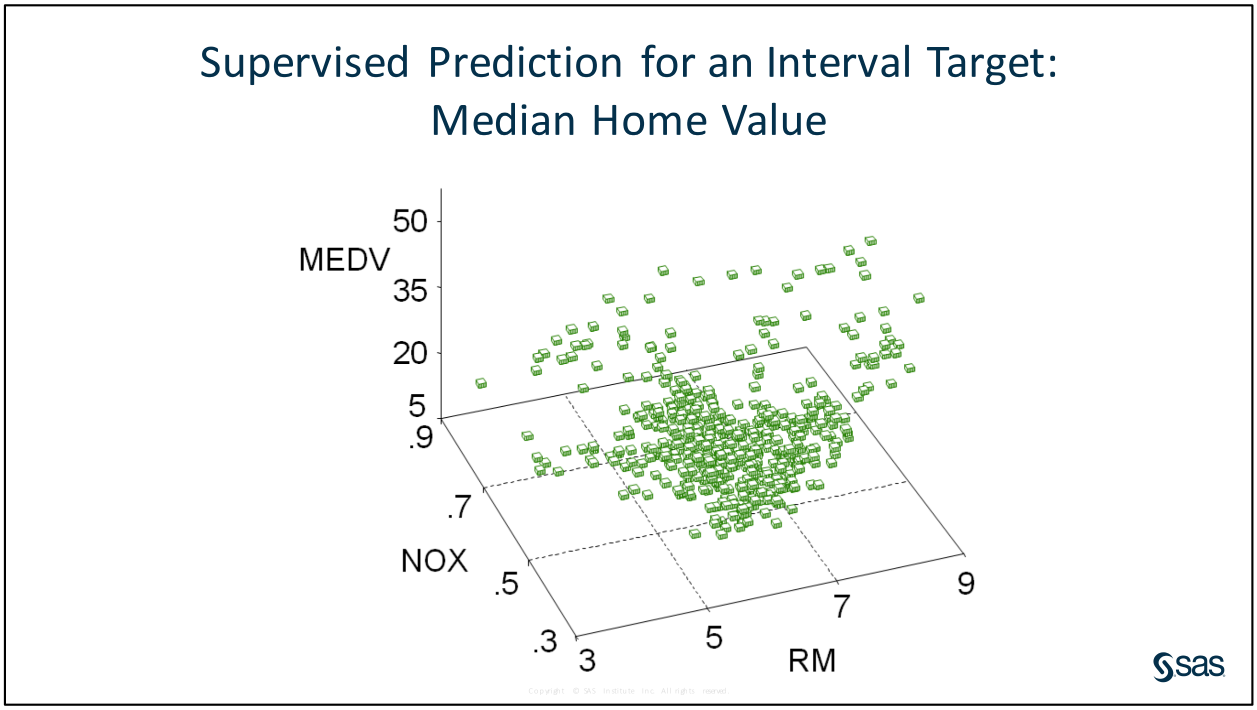
Note: Decision trees can also have multi-way splits where the values of the inputs are partitioned into disjoint ranges. Multi-way splits request more evaluations for the candidate splits, considering all inputs in all n-way splits. For example, in 4-way splits, all possible candidates for 2-way splits, 3-way splits, and 4-way splits are evaluated.



Leaves of a Classification Tree

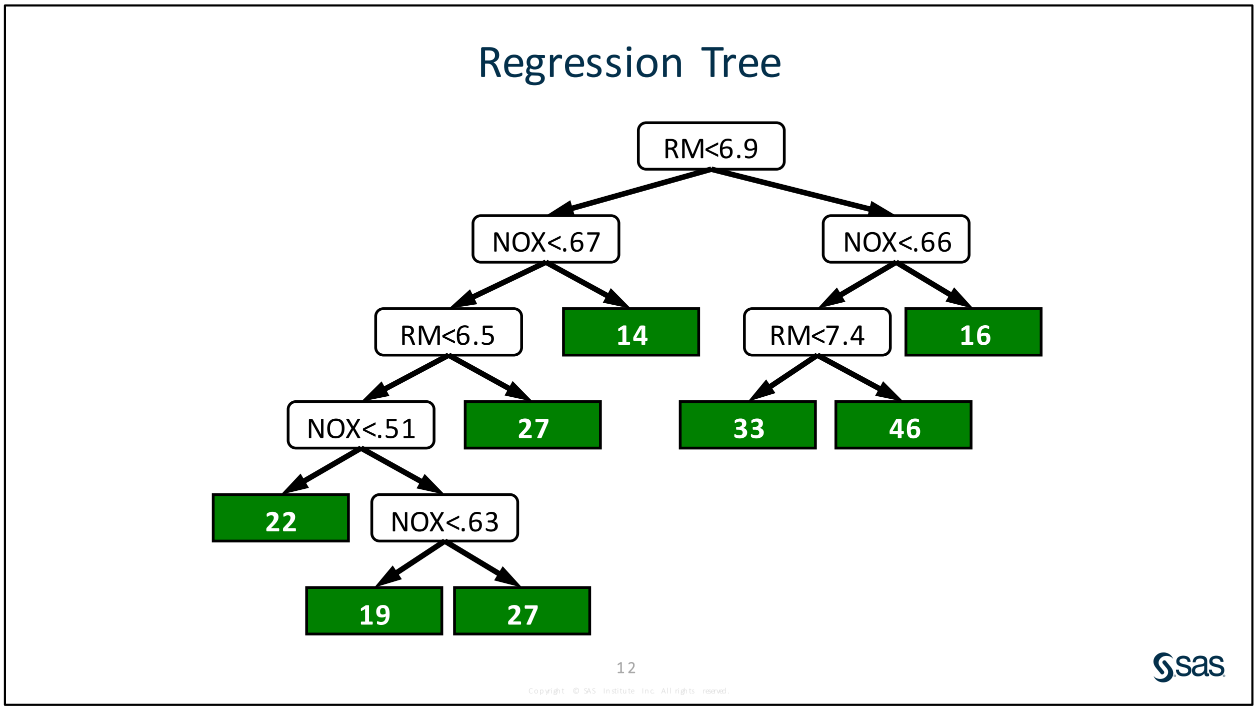
Leaf	Pr(1 |x)	Pr(7 |x)	Pr(9 |x)	Decision
1	.03	.96	.01	7
2	.09	.91	.00	7
3	.56	.44	.00	1
4	.95	.05	.00	1
5	.80	.10	.10	1
6	.64	.09	.27	1
7	.00	.13	.87	9
8	.10	.73	.17	7
9	.78	.01	.21	1
10	.01	.00	.99	9

A classification tree can be thought of as defining several multivariate step functions. Each function corresponds to the posterior probability of a target class.



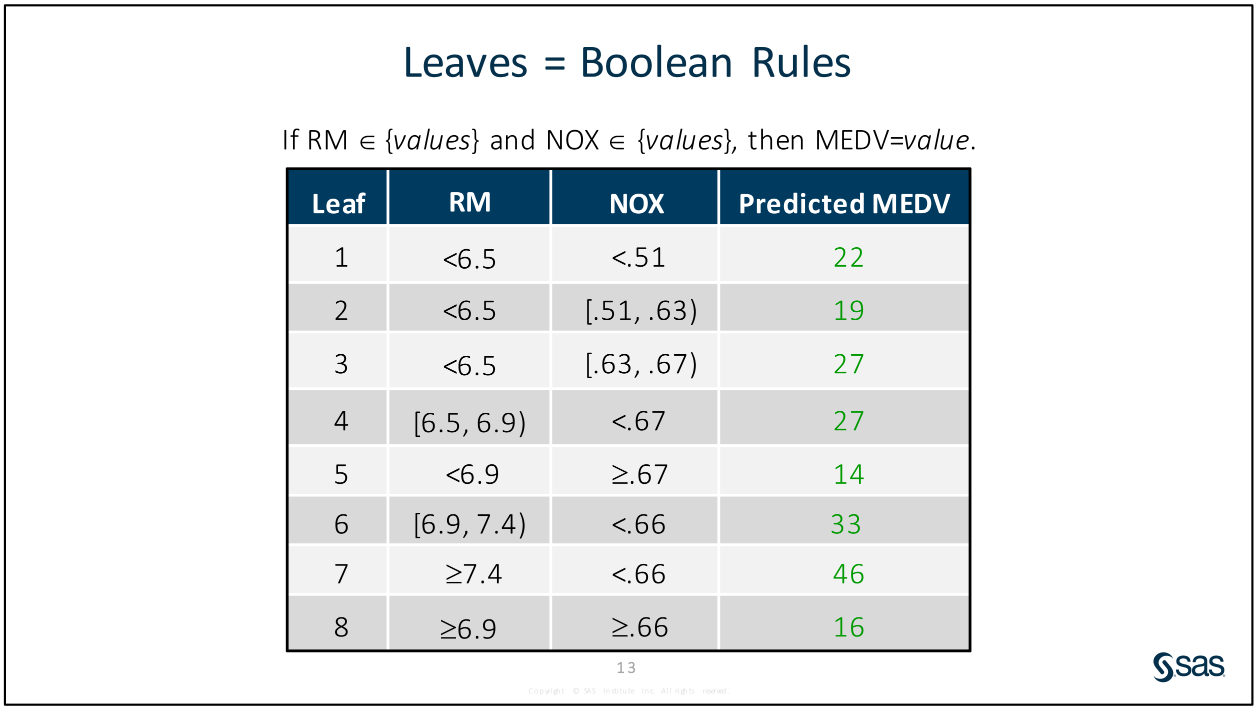
Supervised Prediction for an Interval Target: Median Home Value

The Boston housing data are available from the UCI repository (Blake et al. 1998). The cases are 506 census tracts in Boston. The target is the median home value (**MEDV**). Two of the thirteen inputs are shown: the average number of rooms (**RM**) and the nitrogen oxide concentration in the air (**NOX**).



Regression Tree

When the target is continuous, the model is a called a *regression tree*. The leaves give the predicted value of the target. All cases that reach a particular leaf are assigned the same predicted value.



Leaves = Boolean Rules

If RM ∊ {*values*} and NOX ∊ {*values*}, then MEDV=*value*.

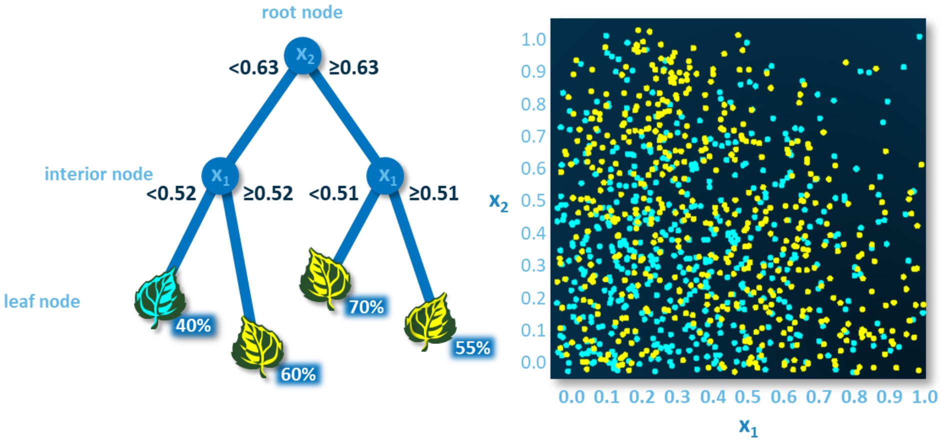
Leaf	RM	NOX	Predicted MEDV
1	<6.5	<.51	22
2	<6.5	[.51, .63)	19
3	<6.5	[.63, .67)	27
4	[6.5, 6.9)	<.67	27
5	<6.9	≥.67	14
6	[6.9, 7.4)	<.66	33
7	≥7.4	<.66	46
8	≥6.9	≥.66	16

The path to each leaf can be expressed as a Boolean rule. The rules take this form:

If the inputs ∊ {*region of the input space*}, then the predicted value = *value*.

The regions of the input space are determined by the split values. For interval-scaled inputs, the boundaries of the regions are perpendicular to the split variables. Consequently, the regions are intersections of subspaces defined by a single splitting variable.

Details: Simple Prediction Illustration



Decision trees provide an excellent introduction to predictive modeling. Decision trees, similar to all modeling methods described in this course, address each of the modeling essentials described in the introduction. Cases are scored using *prediction rules*. A *split-search* algorithm facilitates input selection. Model complexity is addressed by *pruning*.

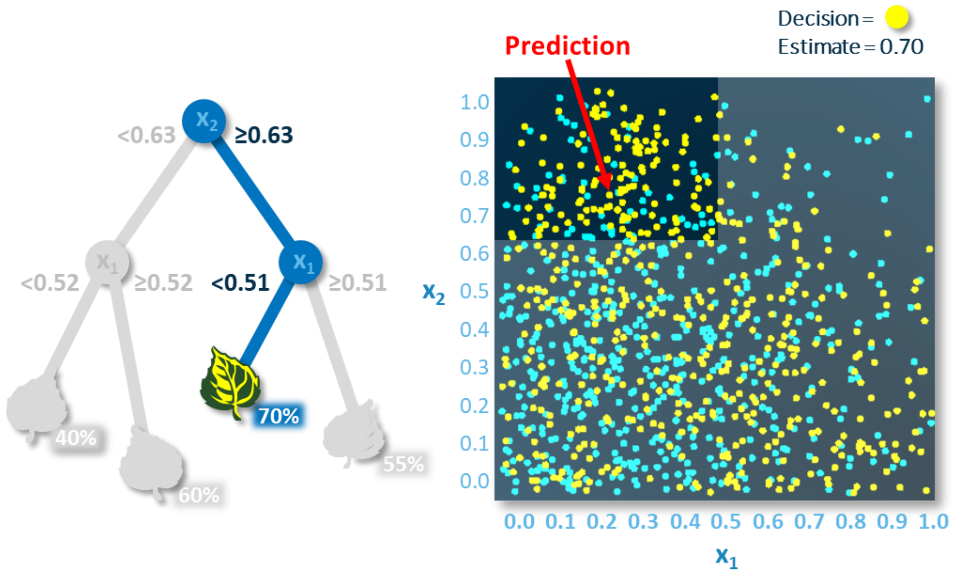
The following simple prediction problem illustrates each of these model essentials:

Consider a data set with two inputs and a binary target. The inputs, x_1 and x_2, locate the case in the unit square. The target outcome is represented by a color: yellow is primary and blue is secondary. The analysis goal is to predict the outcome based on the location in the unit square.

To predict cases, decision trees use rules that involve the values of the input variables.

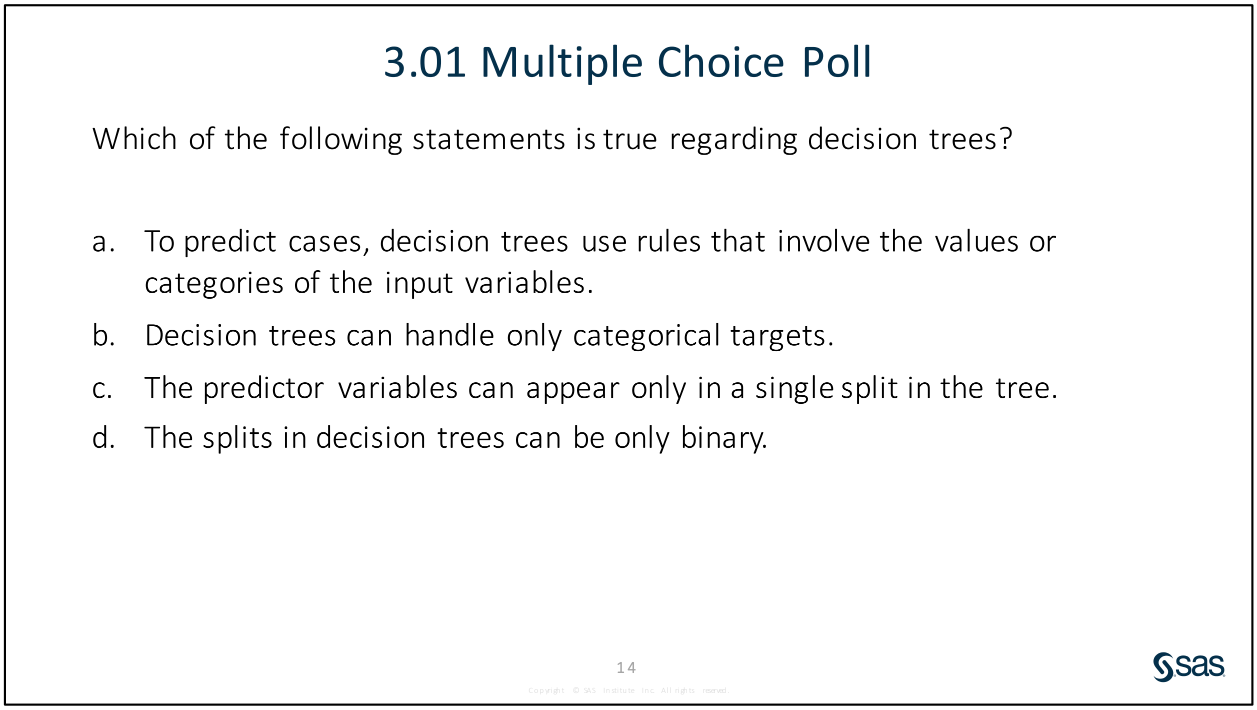
The rules are arranged hierarchically in a tree-like structure with nodes connected by lines. The nodes represent decision rules, and the lines order the rules. The first rule, at the base (top) of the tree, is named the *root node*. Subsequent rules are named *interior nodes*. Nodes with only one connection are *leaf nodes*.

The *depth* of a tree specifies the number of generations of nodes. The root node is generation 0. The children of the root node are the first generation, and so on.



To score a new case, examine the input values and apply the rules defined by the decision tree.

The input values of a new case eventually lead to a single leaf in the tree. A tree leaf provides a decision (for example, classify as yellow) and an estimate (for example, the primary-target proportion).



3.01 Multiple Choice Poll

Which of the following statements is true regarding decision trees?

a. To predict cases, decision trees use rules that involve the values or categories of the input variables.

b. Decision trees can handle only categorical targets.

c. The predictor variables can appear only in a single split in the tree.

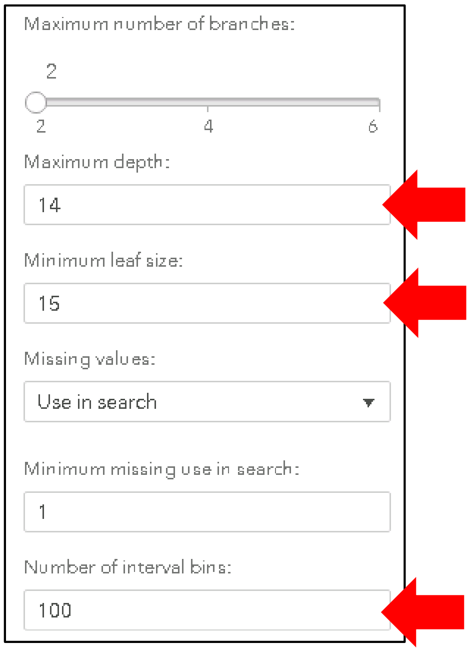
d. The splits in decision trees can be only binary.

14

Improving a Decision Tree Model by Changing the Tree Structure Parameters

In this demonstration, you change the default settings of the Decision Tree node that was just added in the Starter Template pipeline. You modify the tree structure parameters and compare this model performance to the models built earlier in the course.

1. To recall, the previous model, based on the default settings, achieved an average square error of 0.0856 on the VALIDATE partition.
2. Try to improve the model's performance by modifying some of the settings of the Decision Tree model. Expand the **Splitting Options** properties in the properties pane of the Decision Tree node.
3. Increase **Maximum depth** from **10** to **14**.
4. Increase **Minimum leaf size** from **5** to **15**.
5. Increase **Number of interval bins** from **20** to **100**.



Maximum number of branches: 2

Maximum depth: 14

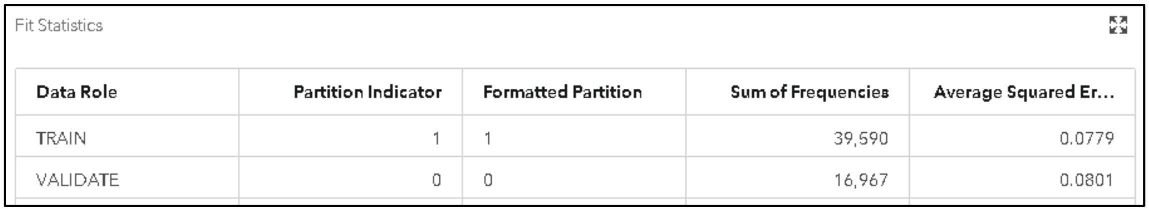
Minimum leaf size: 15

Missing values: Use in search

Minimum missing use in search: 1

Number of interval bins: 100

6. Run the Decision Tree node.
7. Open the results for the Decision Tree node.
8. Click the **Assessment** tab.



Fit Statistics

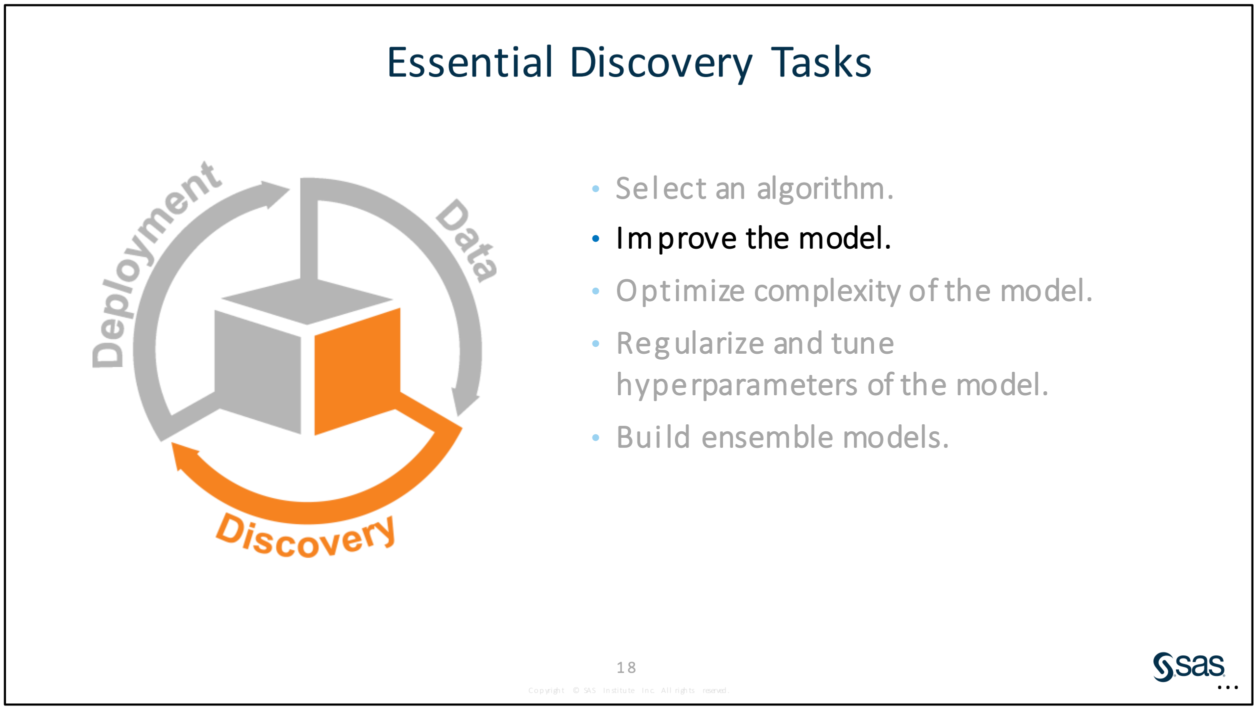
Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.0779
VALIDATE	0	0	16,967	0.0801

The average square error for the tuned Decision Tree model is 0.0801 on the VALIDATE partition. This fit statistic is approximately 6% better than the first model using the default settings.

9. Close the Results window.

End of Demonstration

3.3 Recursive Partitioning

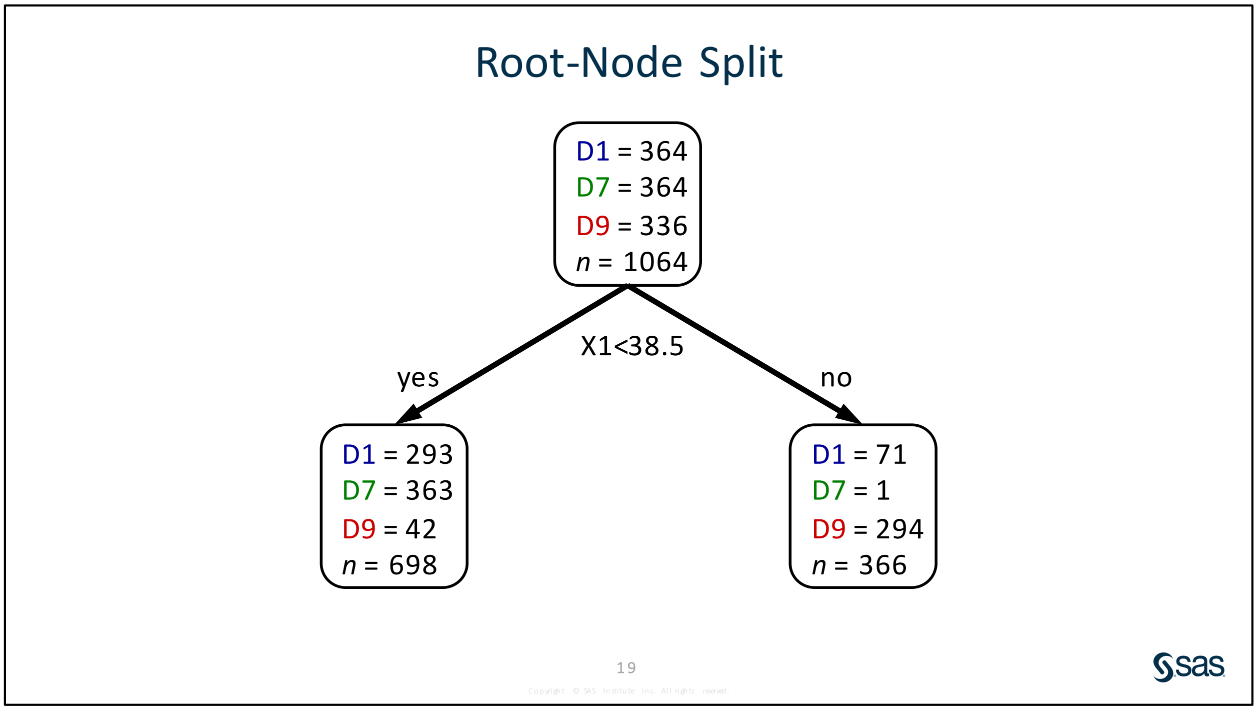


Essential Discovery Tasks

- Select an algorithm.
- **Improve the model.**
- Optimize complexity of the model.
- Regularize and tune hyperparameters of the model.
- Build ensemble models.

18

The most common method to improve decision tree models is by recursive partitioning when we grow a base tree model.



Root-Node Split

D1 = 364
D7 = 364
D9 = 336
n = 1064

X1<38.5

yes

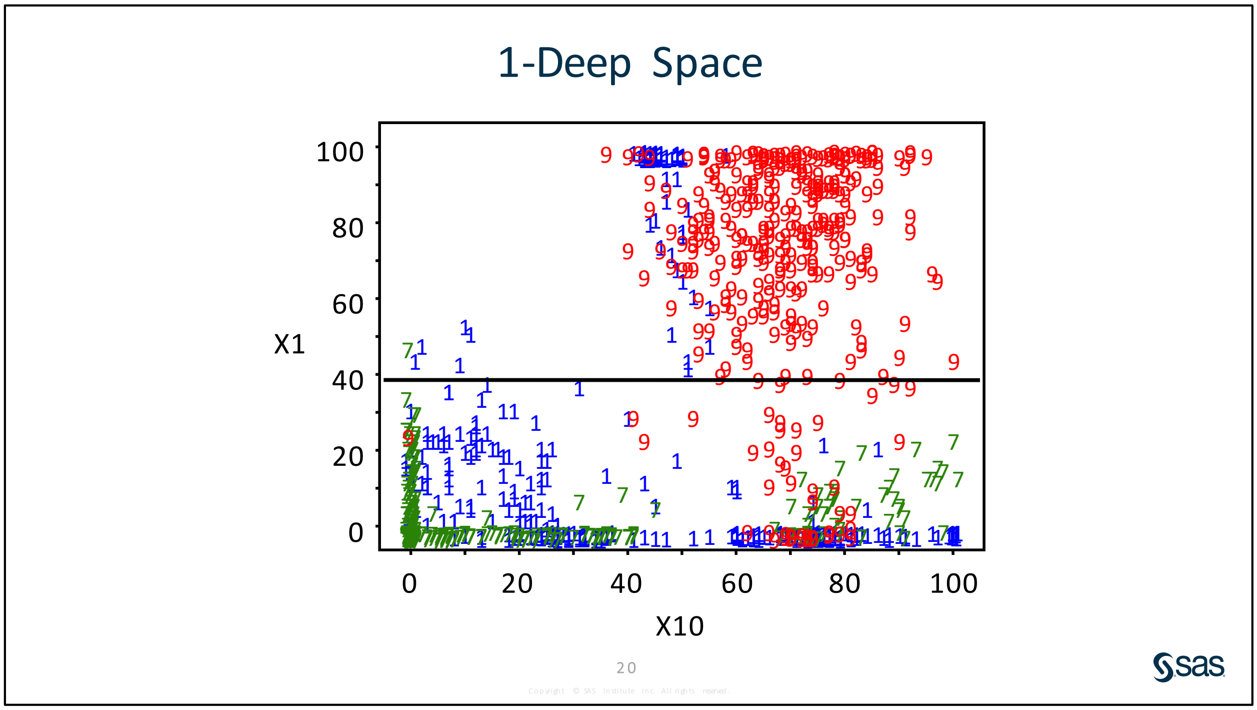
D1 = 293
D7 = 363
D9 = 42
n = 698

no

D1 = 71
D7 = 1
D9 = 294
n = 366

19

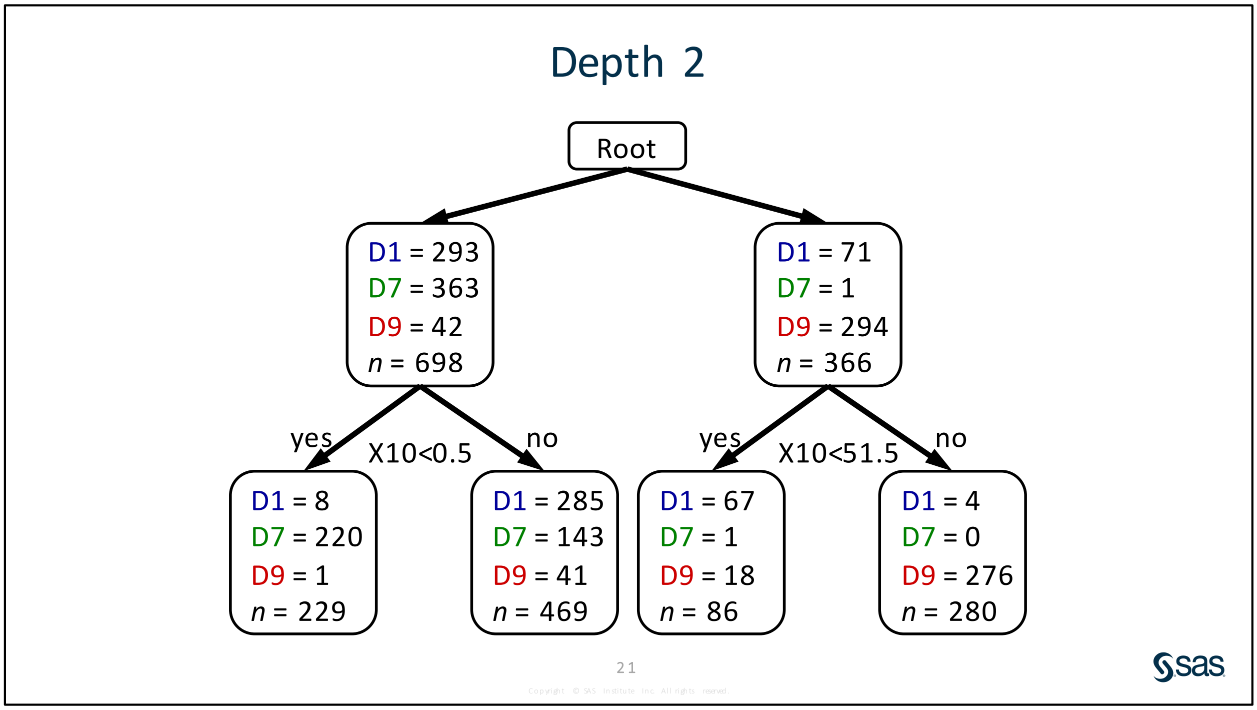
Recursive partitioning is the standard method used to fit decision trees. Recursive partitioning is a top-down, greedy algorithm. A *greedy* algorithm is one that makes locally optimal choices at each step. Starting at the root node, a number of splits that involve a single input are examined. Finding the split point for the root node, is the first step of recursive partitioning. For interval inputs, the splits are disjoint ranges of the input values. For nominal inputs, the splits are disjoint subsets of the input categories. Various split-search strategies can be used to determine the set of candidate splits. A splitting criterion is used to choose the split. The splitting criterion measures the reduction in variability of the target distribution in the child nodes. The goal is to reduce variability and thus increase purity in the child nodes. The cases in the root node are then partitioned according to the selected split.



1-Deep Space

20

The root-node split corresponds to a partition of the input space where the boundary is perpendicular to one input dimension. The result is a tree that has a depth of one, hence the term 1-deep space.



Depth 2

Root

D1 = 293
D7 = 363
D9 = 42
n = 698

D1 = 71
D7 = 1
D9 = 294
n = 366

yes X10<0.5 no

yes X10<51.5 no

D1 = 8
D7 = 220
D9 = 1
n = 229

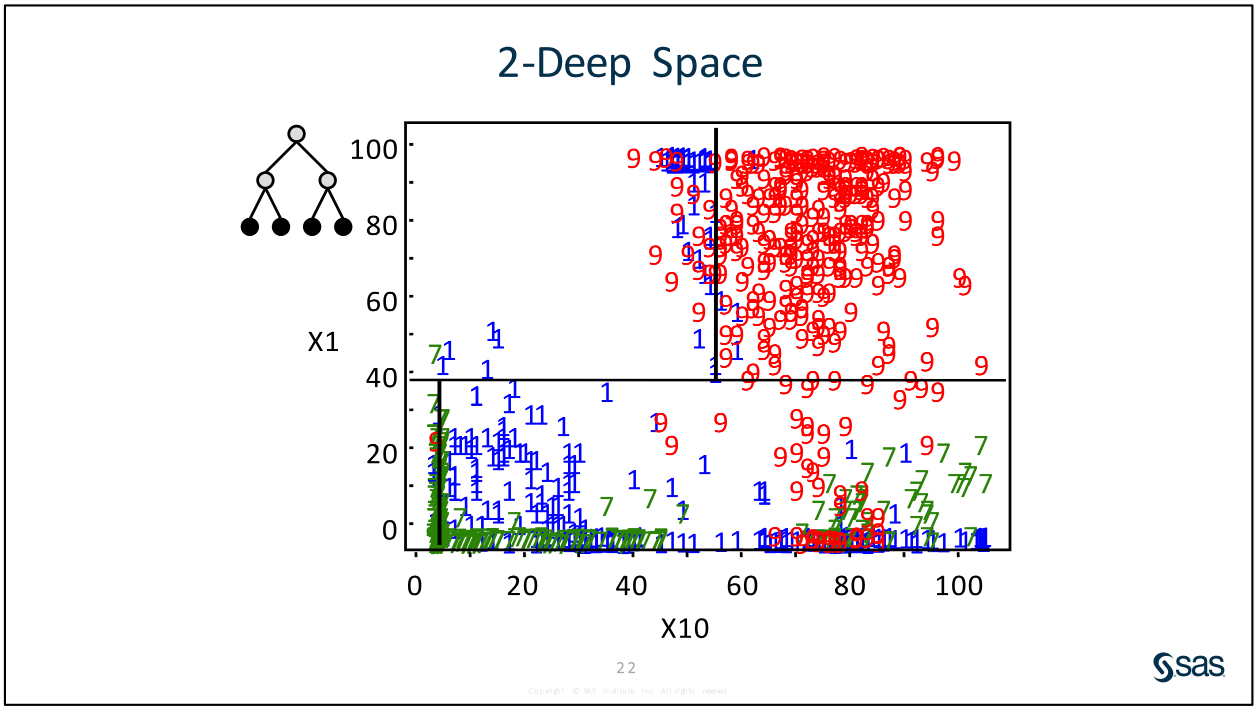
D1 = 285
D7 = 143
D9 = 41
n = 469

D1 = 67
D7 = 1
D9 = 18
n = 86

D1 = 4
D7 = 0
D9 = 276
n = 280

21

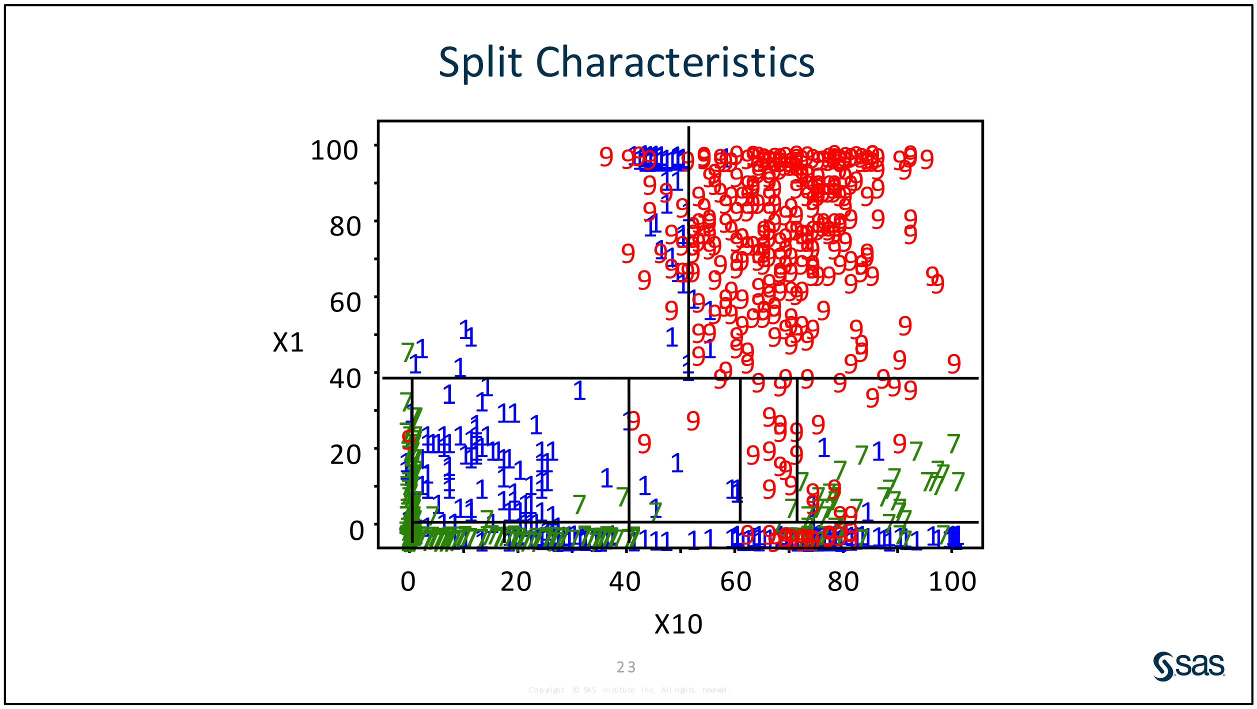
The partitioning is now repeated in each child node as if it were the root node of a new tree. The split selection at a node depends entirely on the cases in that local region of the input space. As the partitioning continues deeper in the tree, the data become more fragmented.



2-Deep Space

22

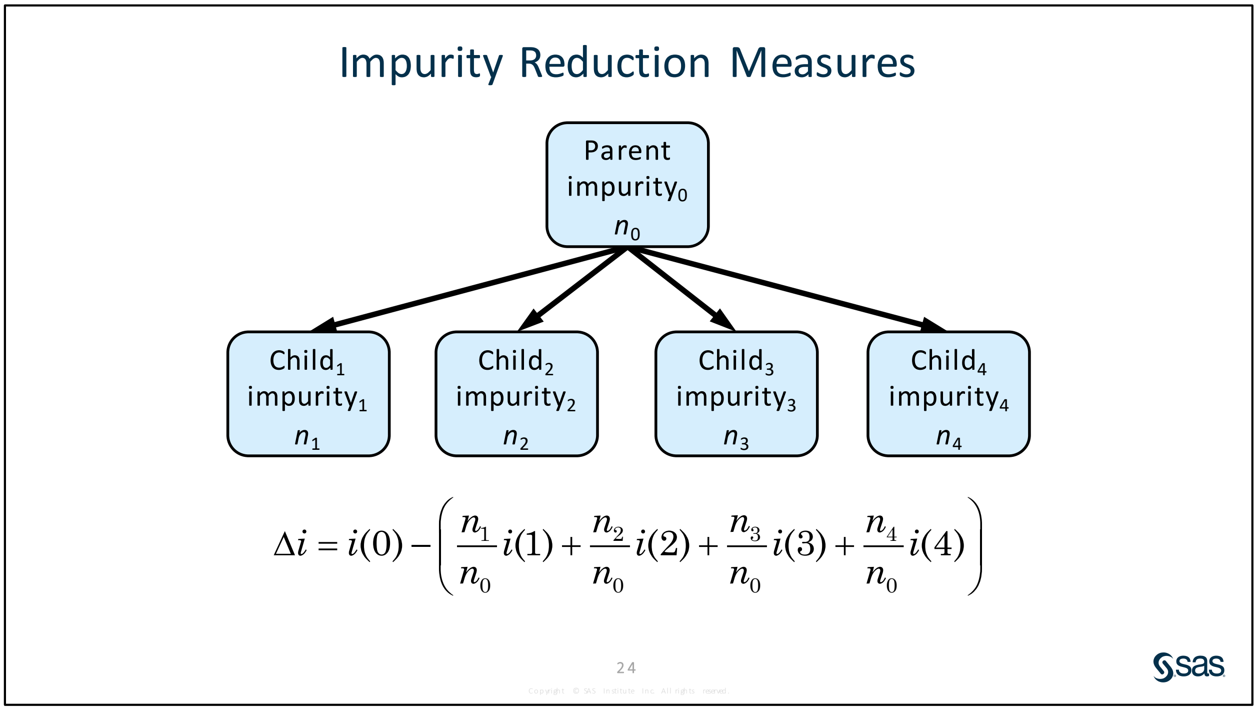
The process is repeated. The depth is governed by stopping rules, which are discussed later.



Split Characteristics

23

It seems reasonable that this greedy algorithm could be improved by incorporating some type of look-ahead or backup. Aside from the computational burden, trees built using limited look-ahead are not shown to be an improvement. In many cases, they produce inferior trees (Murthy and Salzberg 1995).



Impurity Reduction Measures

Parent impurity$_0$ n_0

Child$_1$ impurity$_1$ n_1

Child$_2$ impurity$_2$ n_2

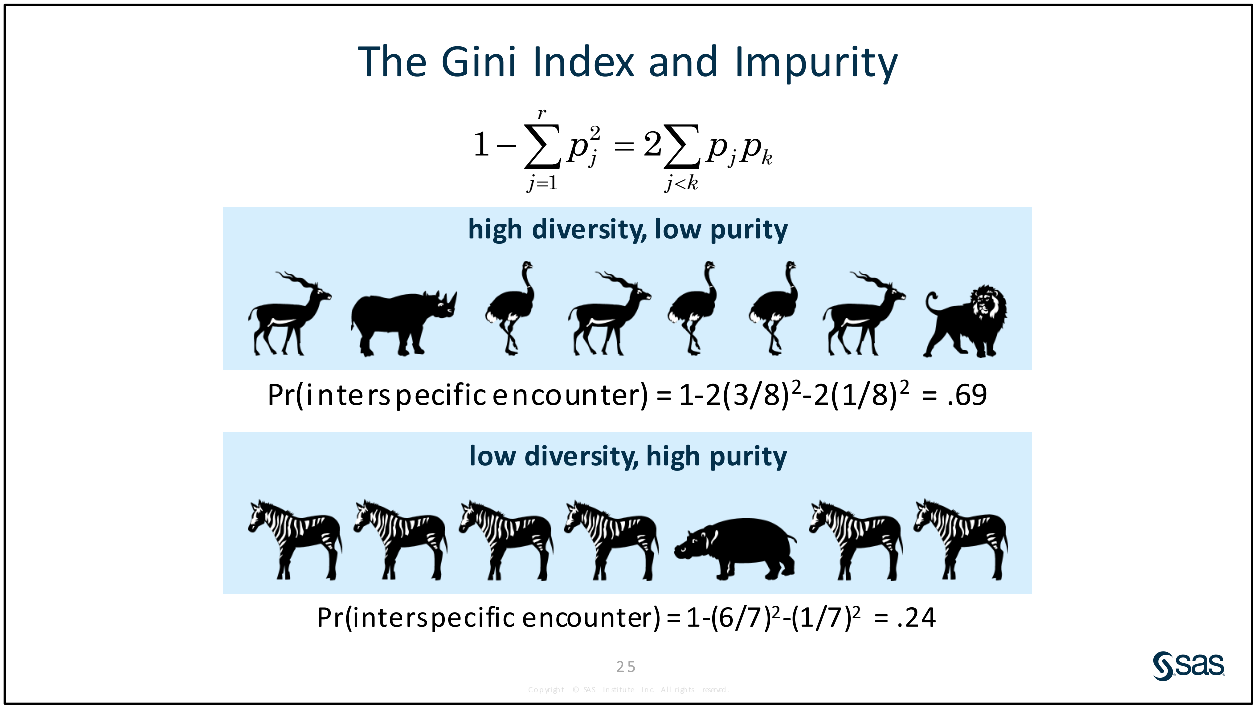
Child$_3$ impurity$_3$ n_3

Child$_4$ impurity$_4$ n_4

$$\Delta i = i(0) - \left(\frac{n_1}{n_0} i(1) + \frac{n_2}{n_0} i(2) + \frac{n_3}{n_0} i(3) + \frac{n_4}{n_0} i(4) \right)$$

24

Let *i*(.) be some measure of within-node impurity, and let *Δi* represent the overall reduction in impurity for the tree. Many splitting criteria (including Gini and entropy) are based on the reduction in node impurity (that is, the reduction of within-node variability) induced by the split.



The Gini Index and Impurity

$$1 - \sum_{j=1}^{r} p_j^2 = 2\sum_{j<k} p_j p_k$$

high diversity, low purity

Pr(interspecific encounter) = 1-2(3/8)^2-2(1/8)^2 = .69

low diversity, high purity

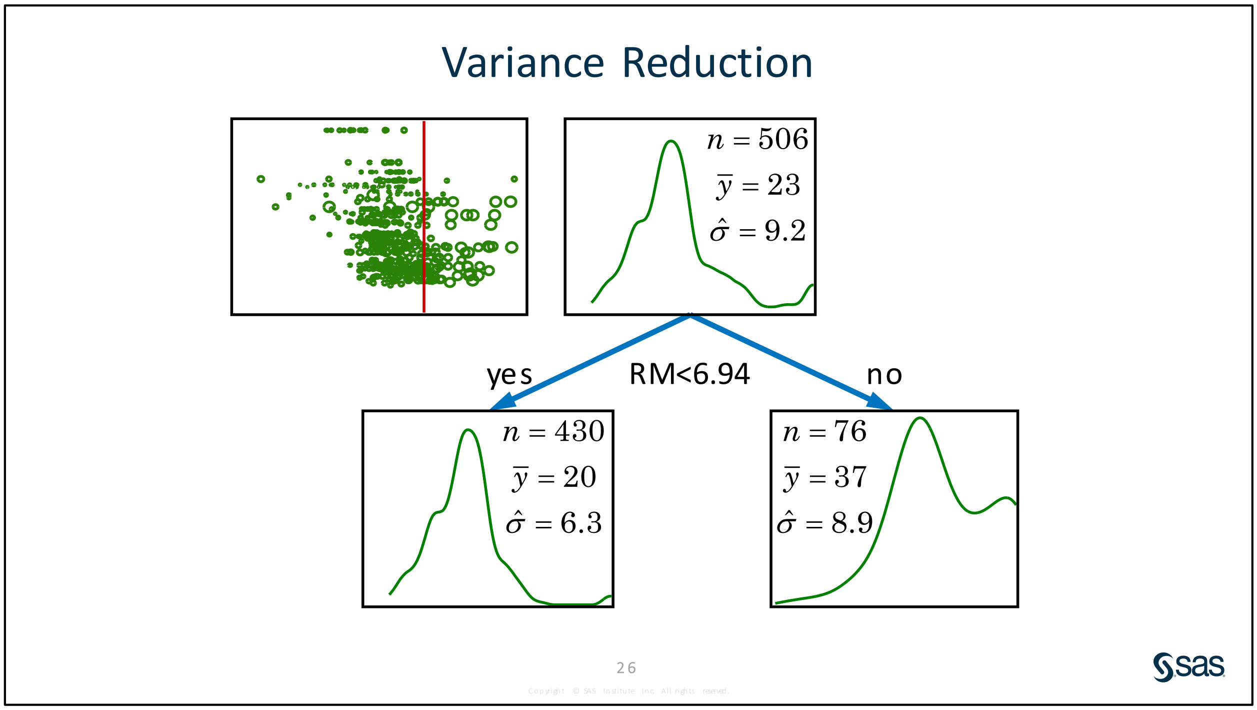
Pr(interspecific encounter) = 1-(6/7)^2-(1/7)^2 = .24

25

The Gini index is a measure of variability for categorical data (developed by the eminent Italian statistician Corrado Gini in 1912). The Gini index can be used as a measure of node impurity where $p_1, p_2, \ldots, p_r$ are the proportions of each target class in a node. The ΔGini splitting criterion was proposed by Breiman et al. (BFOS 1984).

The Gini index can be interpreted as the probability that any two elements of a multi-set, chosen at random (with replacement), are different. A pure node has a Gini index of 0. As the number of evenly distributed classes increases, the Gini index approaches 1.

In mathematical ecology, the Gini index is known as *Simpson's diversity index*. In cryptanalysis, it is 1 minus the ***repeat rate*** (Good, discussion of Patil and Taillie, 1982).



Variance Reduction

When the target has an interval measurement level, splitting criteria can be based on reducing variance of the target in child nodes. Other more robust measures of spread such as the least absolute deviation (LAD) were proposed (BFOS 1984).

Details: Impurity Reduction Measures for Class Targets

X1:	<38.5	≥38.5	ΔGini	Δentropy	Logworth *
1	293	71			
7	363	1	.197	.504	140
9	42	294			

X10:	<0.5	1-41	42-51	≥51.5	ΔGini	Δentropy	Logworth *
1	9	143	65	147			
7	221	88	1	54	.255	.600	172
9	1	4	16	315			

* ΔGini and Δentropy are impurity reduction measures. Logworth is based on the χ^2 from an *n-way* table. For all, higher is better.

After a set of candidate splits is determined, a splitting criterion is used to determine the best one. In some situations, the worth of a split is obvious. If target distributions are the same in the child nodes as they are in the parent node, then no improvement was made, and the split is worthless. In contrast, if a split results in pure children, then the split is definitely the best.

In classification trees, the three most well-known splitting criteria are based on the Gini index (BFOS 1984), entropy (Quinlan 1993), and the chi-square test (Kass 1980). Well-known algorithms and software products associated with these three splitting criteria are CART (classification and regression tree); C5.0 (developed by the machine learning researcher Quinlin); and the CHAID algorithm (chi-squared automatic interaction detection).

Details: Impurity Reduction Measures for Interval Targets

Regression trees endeavor to partition the input space into segments where the target values are alike. (That is, each segment or node has low variability.) All target values would be equal in a pure node. In other words, the variance of the target would be zero within a pure node.

The split-search considerations and the *p*-value adjustments are the same as with classification trees. However, the appropriate splitting criteria are different. The default splitting criterion for a regression tree is change in response variance. CHAID and F Statistic are also available as splitting methods.

Entropy and the Gini index are measures of variability of nominal variables. When the target distribution is continuous, the sample variance is the obvious measure of impurity. (Morgan and Sonquist 1963, BFOS 1984)

$$i(t) = \frac{1}{n_t}\sum_{j=1}^{n_t}\left(y_{jt} - \bar{y}_t\right)^2$$

Note: The denominator is n_i, not n_i–1. This is the MLE and not the usual unbiased estimate of sample variance.

The *F* test can be used analogously to the chi-square test for regression trees. A split at a node can be thought of as a one-way analysis of variance where the *B* branches are the *B* treatments. Let $\bar{y}_{i\cdot} = \frac{1}{n_i}\sum_{j=1}^{n_i} y_{ij}$ be the mean of the target in each node and $\bar{y}_{\cdot\cdot}$ be the mean in the root node (the overall mean). The between-node sum of squares (SS$_{\text{between}}$) is a measure of the distance between the node means and the overall mean. The within-node sum of squares (SS$_{\text{within}}$) measures the variability within a node. Large values of the *F* statistic indicate departures from the null hypothesis that all the node means are equal. When the target values, conditional on the inputs, are independently, normally distributed with constant variance, then the *F* statistic follows an *F* distribution with *B* – 1 and *n* – *B* degrees of freedom. The *p*-value of the test is used in the same way as the *p*-value for a chi-square test for classification trees.

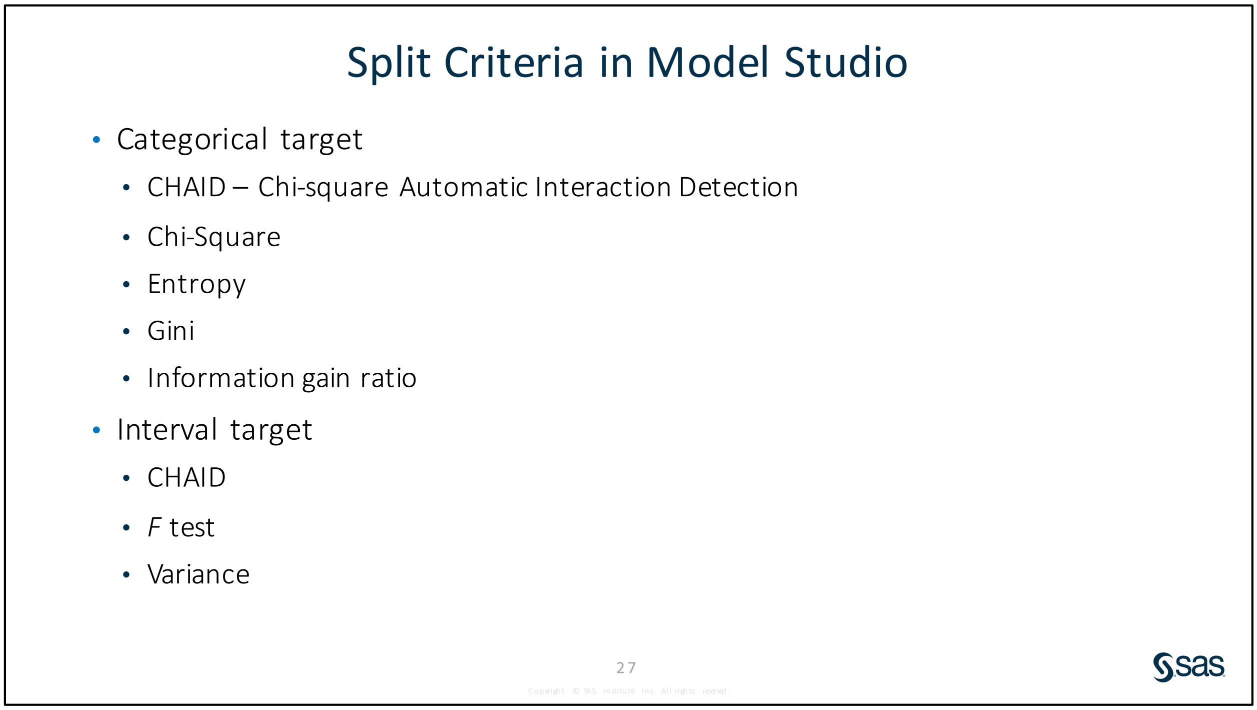
The total sum of squares (SS$_{\text{total}}$) can be considered fixed with regard to comparing splits at a particular node. Thus, it follows from the ANOVA identity

$$\text{SS}_{\text{total}} = \text{SS}_{\text{between}} + \text{SS}_{\text{within}}$$

that the *F* test statistic can be thought of as either maximizing the differences between the node means or reducing the within-node variance. This latter interpretation indicates the equivalency between the *F* statistic and the reduction in impurity (variance) splitting criterion.

$$\Delta\,\text{var} = \frac{\text{SS}_{\text{total}}}{n} - \sum_{i=1}^{B}\left(\frac{n_i}{n}\right)\left(\frac{\text{SS}_i}{n_i}\right) = \frac{1}{n}\left(\text{SS}_{\text{total}} - \text{SS}_{\text{within}}\right) = \frac{\text{SS}_{\text{between}}}{n}$$

Thus, using Δvariance is equivalent to ***not*** adjusting the *F* test for degrees of freedom (number of branches).



Split Criteria in Model Studio

- Categorical target
  - CHAID – Chi-square Automatic Interaction Detection
  - Chi-Square
  - Entropy
  - Gini
  - Information gain ratio
- Interval target
  - CHAID
  - *F* test
  - Variance

27

In summary, these are the options in Model Studio to grow a decision tree.

For categorical responses, the available criteria are CHAID, CHISQUARE, ENTROPY, GINI, and IGR (information gain ratio). The default is IGR. For continuous responses, the available criteria are CHAID, FTEST, and VARIANCE. The default is VARIANCE.

CHAID uses the value of a chi-square statistic for a classification tree or an *F* statistic for a regression tree. Based on the significance level, the value of the chi-square of *F* statistic is used to merge similar levels of the predictor variable until the number of children in the proposed split reaches the number specified as the maximum possible branches. The *p*-values for the final split determine the variable on which to split.

Split criteria using the *p*-value (Chi-square, CHAID of F Test) can request a Bonferroni adjustment to the *p*-value for a variable after the split has been determined.

CHISQUARE uses a chi-square statistic to split each variable and then uses the *p*-values that correspond to the resulting splits to determine the splitting variable.

ENTROPY uses the gain in information or the decrease in entropy to split each variable and then to determine the split. A minimum of decrease in entropy or increase in information gain ration can be specified.

GINI uses the decrease in the Gini index to split each variable and then to determine the split.

IGR uses the entropy metric to split each variable and then uses the information gain ratio to determine the split.

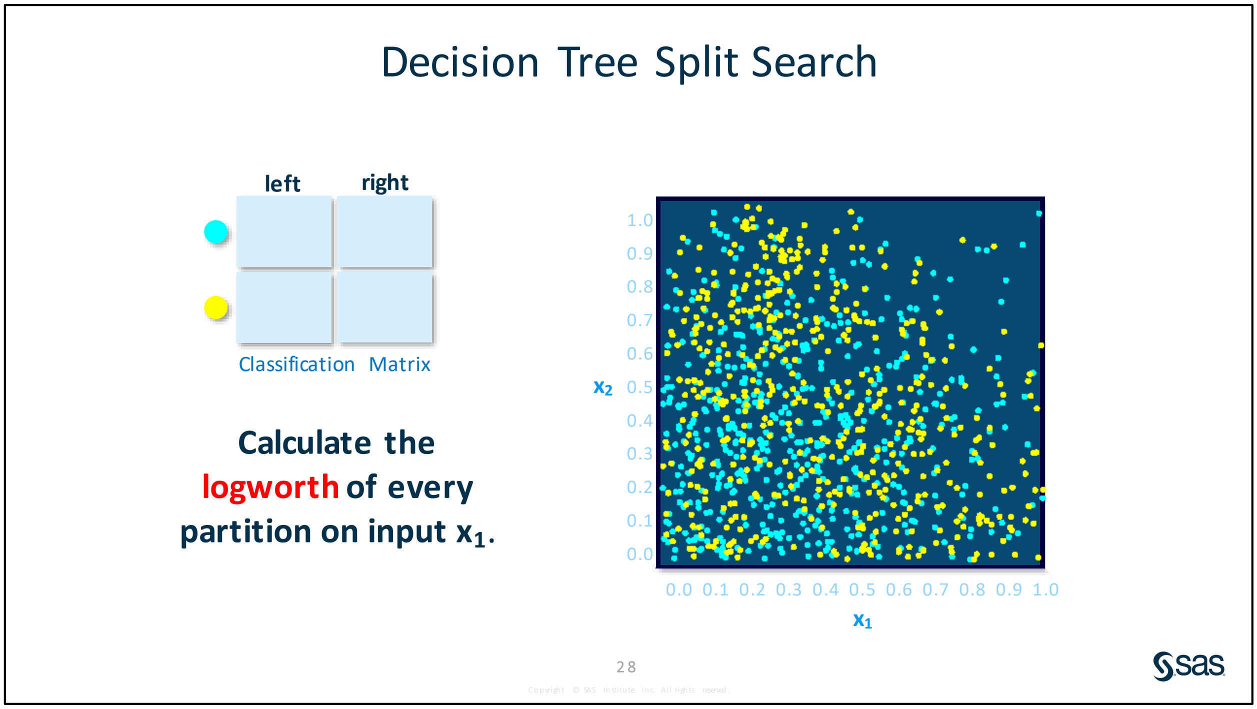
For continuous responses the available criteria are CHAID, FTEST and VARIANCE.

CHAID is described above.

FTEST uses an *F* statistic to split each variable and then uses the resulting *p*-value to determine the split point.

A Bonferroni adjustment can be applied to both CHAID and FTEST criteria.

VARIANCE uses the change in response variance to split each variable and then to determine the split.



Decision Tree Split Search

Calculate the logworth of every partition on input x_1.

28

To select useful inputs, trees use a *split-search* algorithm. Decision trees confront the curse of dimensionality by ignoring irrelevant inputs.

Note: Curiously, trees have no built-in method for ignoring redundant inputs. Because trees can be trained quickly and have a simple structure, this is usually not an issue for model creation. However, it can be an issue for model deployment, in that trees might somewhat arbitrarily select from a set of correlated inputs. To avoid this problem, you must use an algorithm that is external to the tree to manage input redundancy.

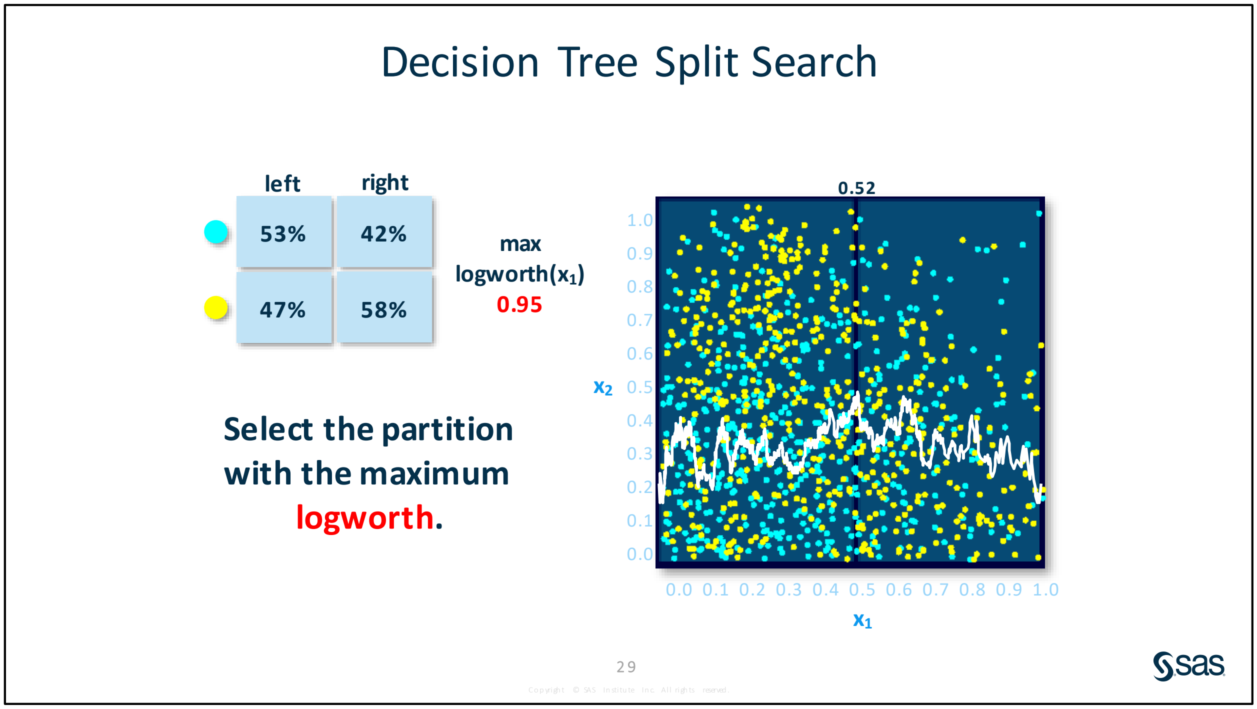
Understanding the default algorithm for building trees enables you to better use SAS Visual Data Mining and Machine Learning to build a tree and interpret your results. The description presented here assumes a binary target, but the algorithm for interval targets is similar. (The algorithm for categorical targets with more than two outcomes is more complicated and is not discussed.)

The first part of the algorithm is called the *split search*. The split search starts by selecting an input for partitioning the available training data. If the measurement scale of the selected input is *interval*, each unique value serves as a potential split point for the data. If the input is *categorical*, the average value of the target is taken within each categorical input level. The averages serve the same role as the unique interval input values in the discussion that follows.

For a selected input and fixed split point, two groups are generated. Cases with input values less than the split point are said to *branch left*. Cases with input values greater than the split point are said to *branch right*. The groups, combined with the target outcomes, form a 2x2 contingency table with columns specifying branch direction (left or right) and rows specifying target value (0 or 1). A Pearson chi-squared statistic is used to quantify the independence of counts in the table's columns. Large values for the chi-squared statistic suggest that the proportion of zeros and ones in the left branch is different from the proportion in the right branch. A large difference in outcome proportions indicates a good split.

Because the Pearson chi-squared statistic can be applied to the case of multiway splits and multi-outcome targets, the statistic is converted to a probability value, or *p*-value. The *p*-value indicates the likelihood of obtaining the observed value of the statistic assuming identical target proportions in each branch direction. For large data sets, these *p*-values can be very close to zero. For this reason, the quality of a split is reported by *logworth* = -*log*(chi-squared *p*-value).

Note: At least one logworth must exceed a threshold for a split to occur with that input. By default, this threshold corresponds to a chi-squared *p*-value of 0.20 or a logworth of approximately 0.7.



Decision Tree Split Search

	left	right
●	53%	42%
●	47%	58%

max logworth(x_1) 0.95

Select the partition with the maximum logworth.

The best split for an input is the split that yields the highest logworth.

Several peripheral factors make the split search somewhat more complicated than what is described above.

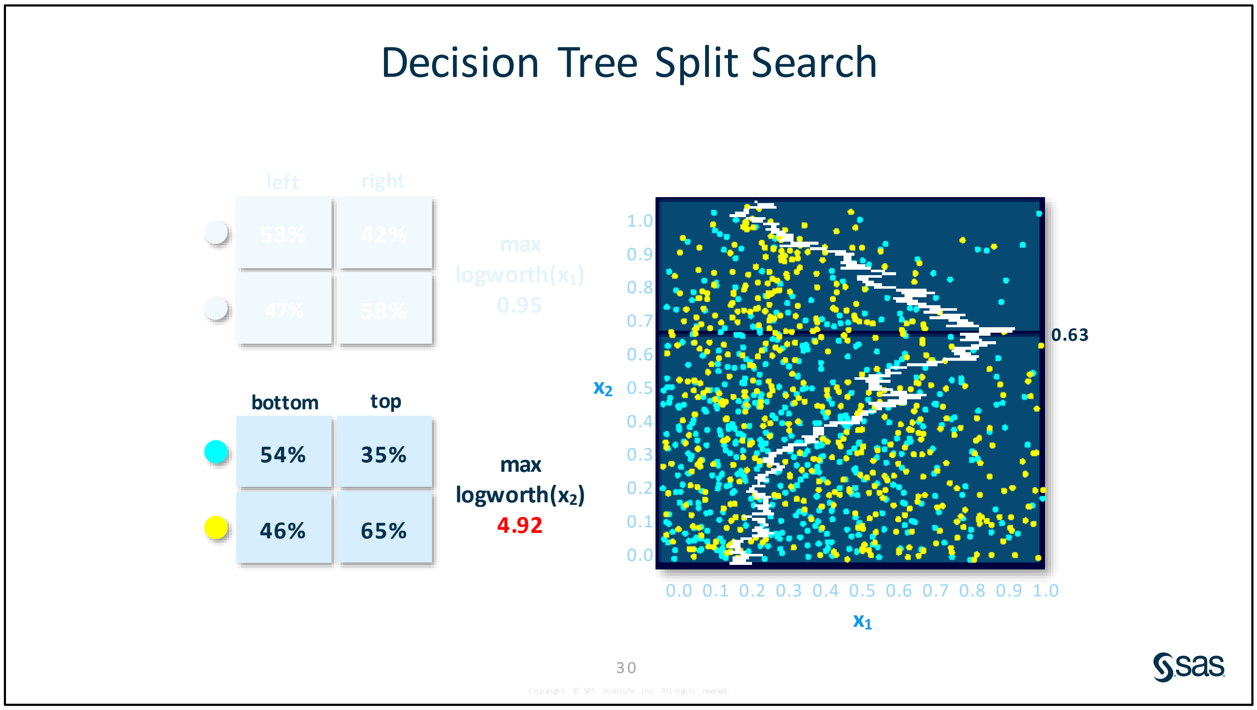
First, the tree algorithm settings disallow certain partitions of the data. Settings, such as the minimum number of observations required for a split search and the minimum number of observations in a leaf, force a minimum number of cases in a split partition. This minimum number of cases reduces the number of potential partitions for each input in the split search.

Second, when you test for the independence of column categories in a contingency table, it is possible to obtain significant (large) values of the chi-squared statistic even when there are no differences in the true, underlying proportions between split branches. In other words, if there are many ways to split the variable that labels the rows of the table (and thus many Chi-square tables and tests), then you are likely to get at least one with a very small *p*-value even when the variable has no true effect. As the number of possible split points increases, the likelihood of obtaining significant values also increases. In this way, an input with a multitude of unique input values has a greater chance of accidentally having a large logworth than an input with only a few distinct input values.

Statisticians face a similar problem when they combine the results from multiple statistical tests. As the number of tests increases, the chance of a false positive result likewise increases. To maintain overall confidence in the statistical findings, statisticians inflate the *p*-values of each test by a factor equal to the number of tests being conducted. If an inflated *p*-value shows a significant result, then the significance of the overall results is assured. This type of *p*-value adjustment is known as a *Bonferroni correction*.

Because each split point corresponds to a statistical test, Bonferroni corrections are automatically applied to the logworth calculations for an input. These corrections, also called *Bonferroni adjustments*, penalize inputs with many split points by reducing the logworth of a split by an amount equal to the log of the number of distinct input values. This is equivalent to the Bonferroni correction because subtracting this constant from logworth is equivalent to multiplying the corresponding chi-squared *p*-value by the number of split points. The adjustment enables a fairer comparison of inputs with many and few levels later in the split-search algorithm.

Third, for inputs with missing values, two sets of Bonferroni-adjusted logworths are generated. For the first set, cases with missing input values are included in the left branch of the contingency table and logworths are calculated. For the second set of logworths, missing value cases are moved to the right branch. The best split is then selected from the set of possible splits with the missing values in the left and right branches, respectively.



Decision Tree Split Search

	left	right
○	53%	42%
○	47%	58%

max logworth(x_1) 0.95

	bottom	top
●	54%	35%
●	46%	65%

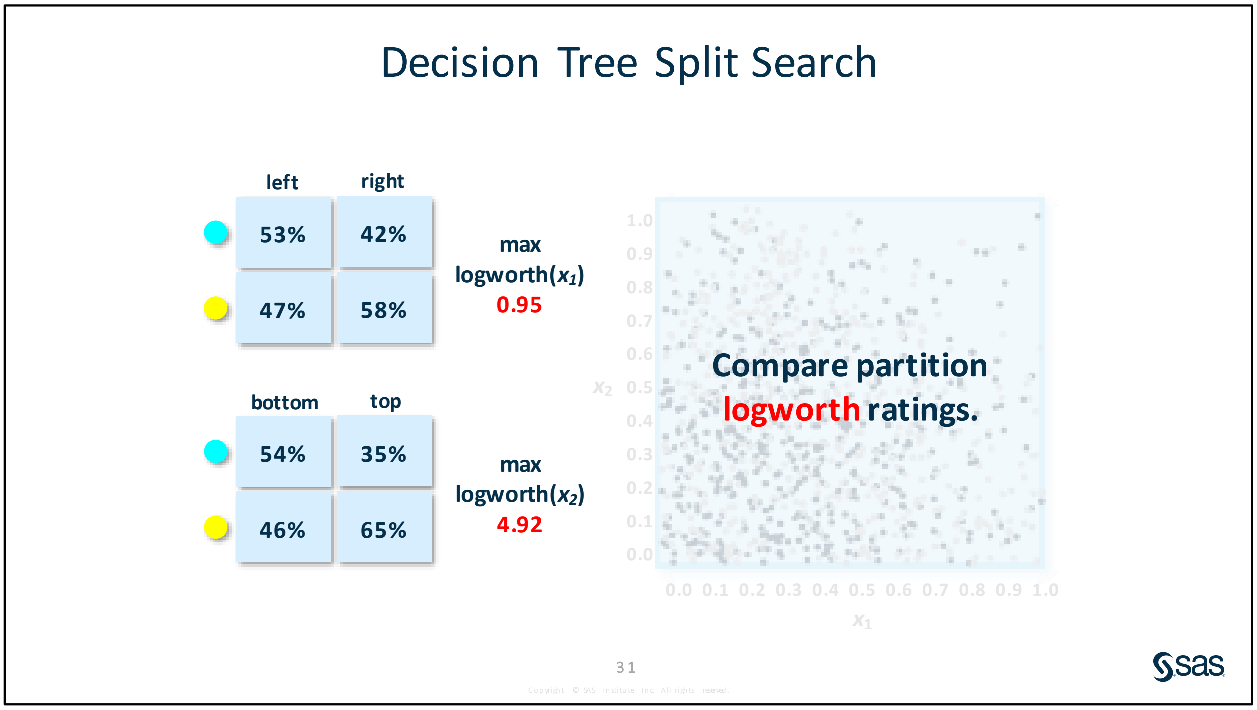
max logworth(x_2) 4.92

30

Copyright © SAS Institute Inc. All rights reserved.

The partitioning process is repeated for every input in the training data. Inputs whose adjusted logworth fails to exceed the threshold are excluded from consideration.

Again, the optimal split for the next input considered is the one that maximizes the logworth function for that input.



Decision Tree Split Search

	left	right
●	53%	42%
●	47%	58%

max logworth(x_1) 0.95

	bottom	top
●	54%	35%
●	46%	65%

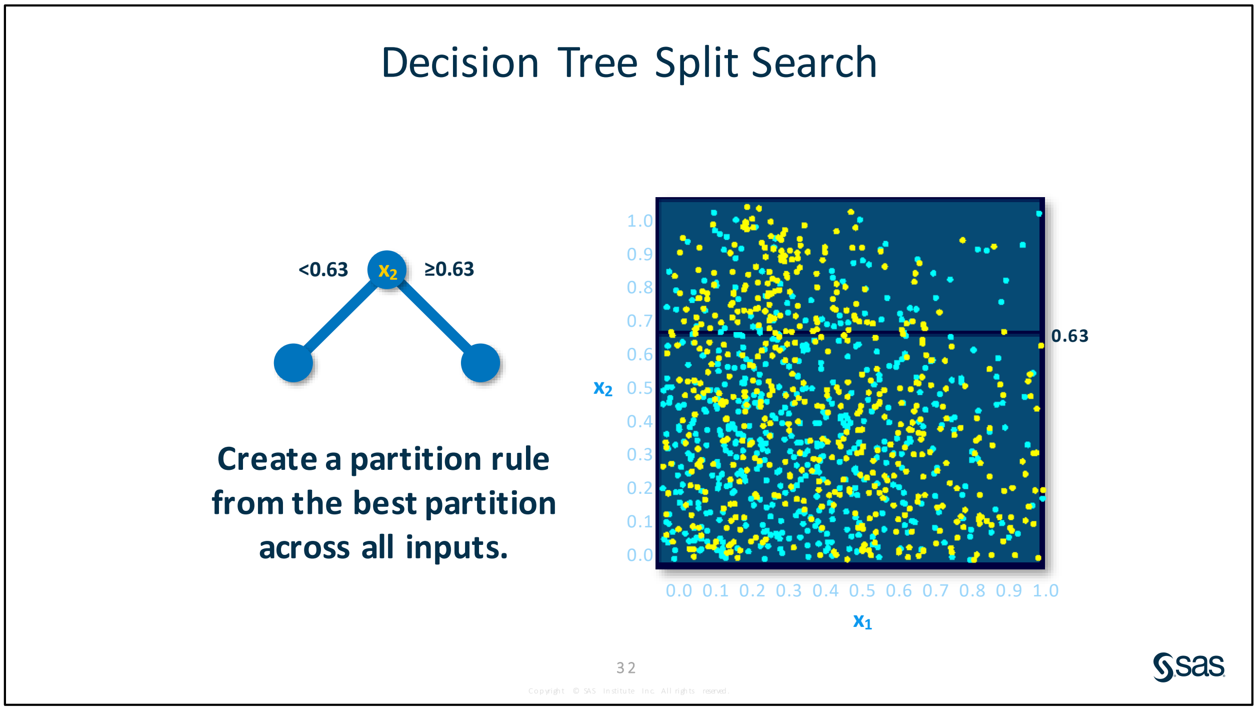
max logworth(x_2) 4.92

Compare partition logworth ratings.

31

Copyright © SAS Institute Inc. All rights reserved.

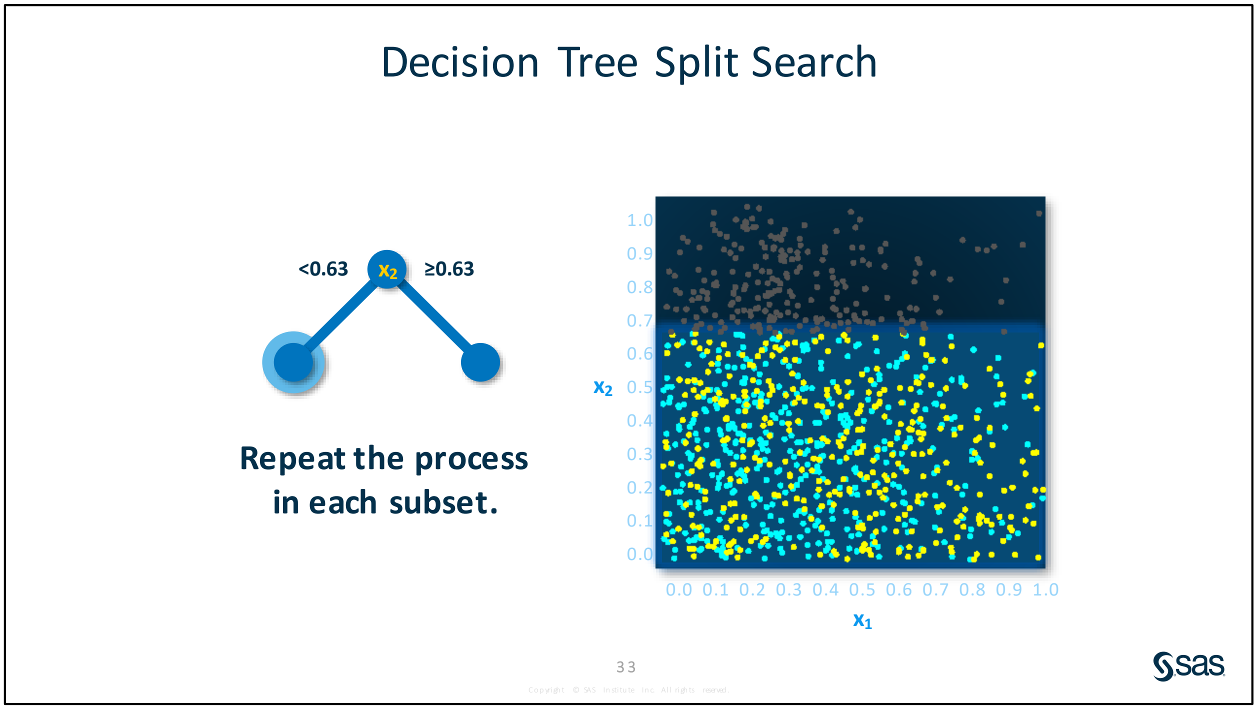
After you determine the best split for every input, the tree algorithm compares each best split's corresponding logworth. The split with the highest adjusted logworth is deemed best.



Decision Tree Split Search

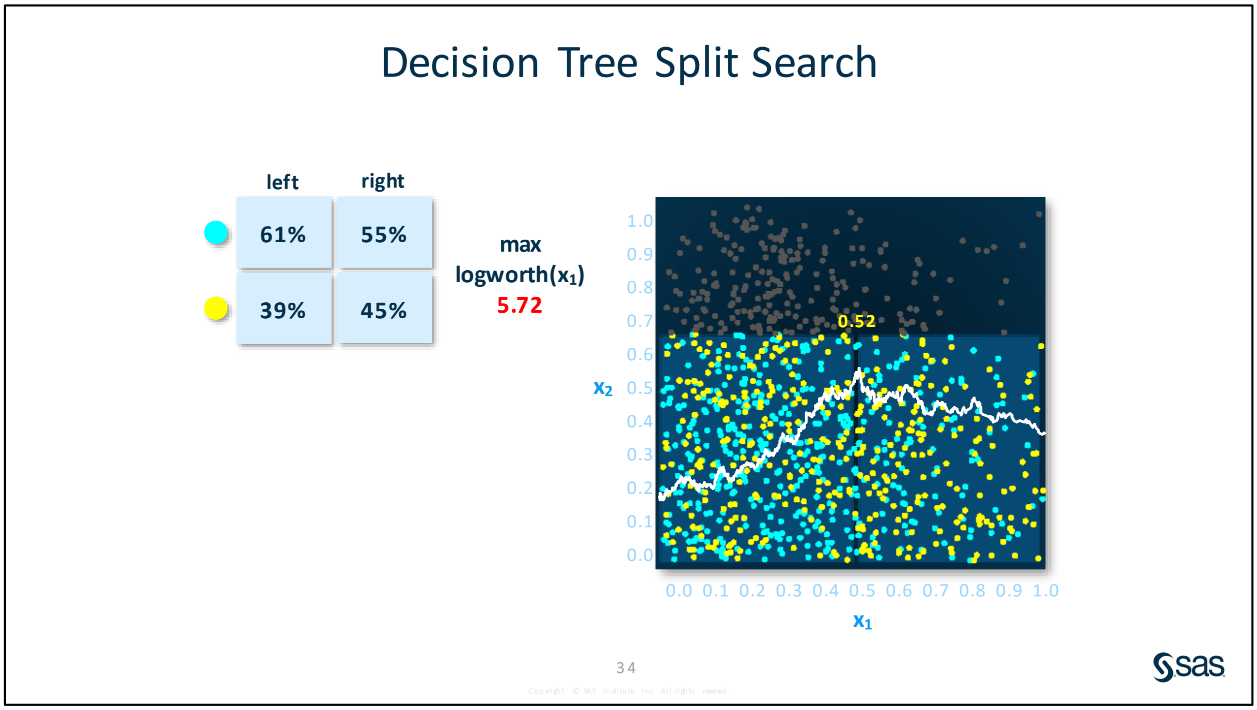
Create a partition rule from the best partition across all inputs.

The training data are partitioned using the best split rule.



Decision Tree Split Search

Repeat the process in each subset.

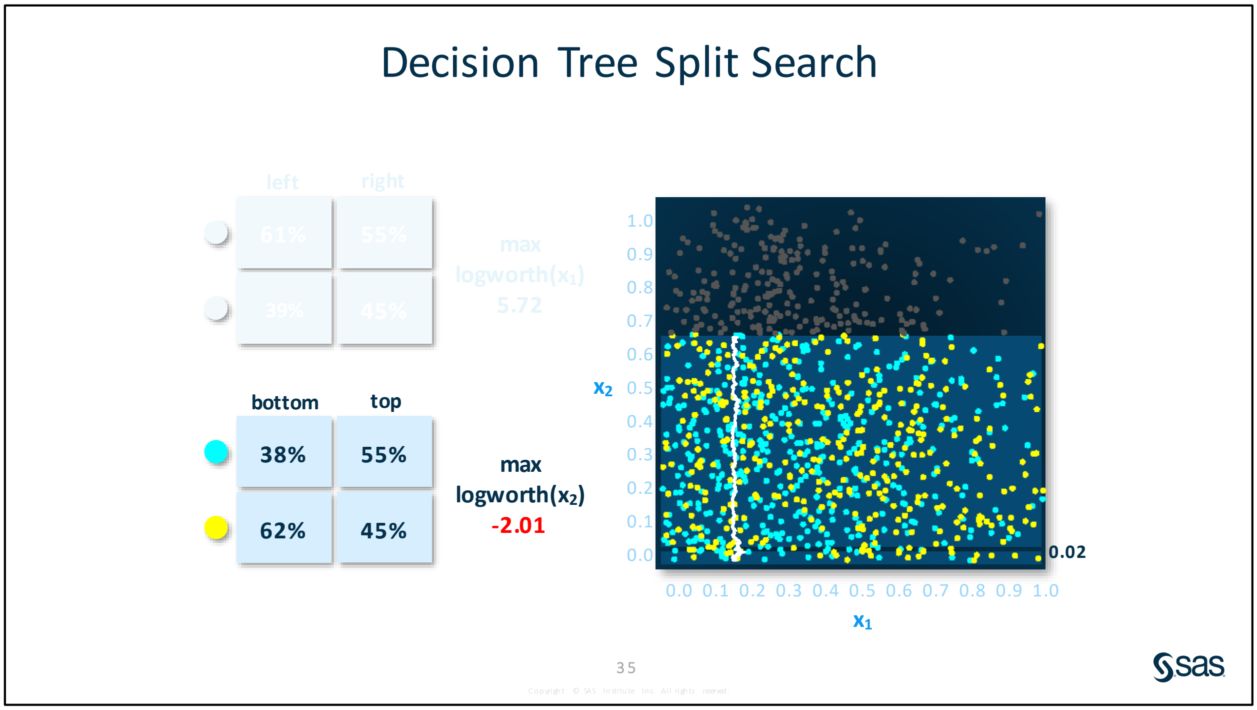


Decision Tree Split Search

	left	right
●	61%	55%
●	39%	45%

max logworth(x_1)
5.72

34



Decision Tree Split Search

	left	right
●	61%	55%
●	39%	45%

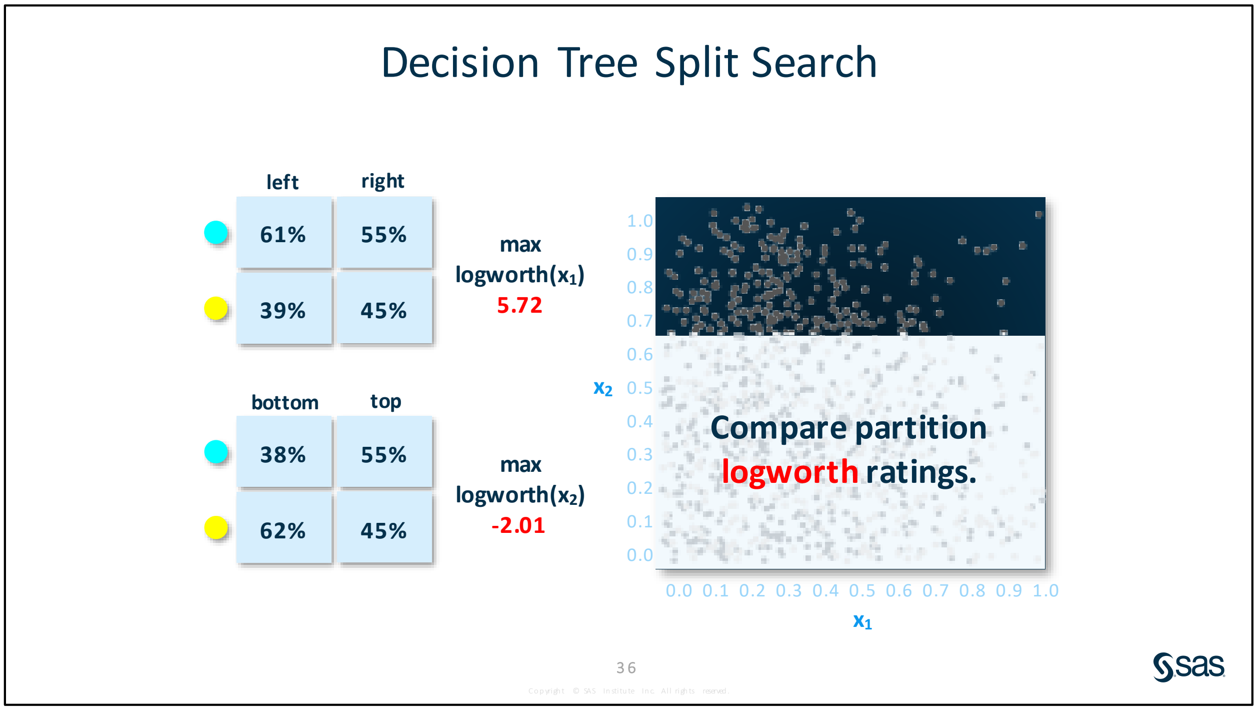
max logworth(x_1)
5.72

	bottom	top
●	38%	55%
●	62%	45%

max logworth(x_2)
-2.01

35

The logworth of the x_2 split is negative. This might seem surprising, but it results from several adjustments made to the logworth calculation. (The Bonferroni adjustment was described previously. Another, called the *depth adjustment*, is outlined in a self-study section.)



Decision Tree Split Search

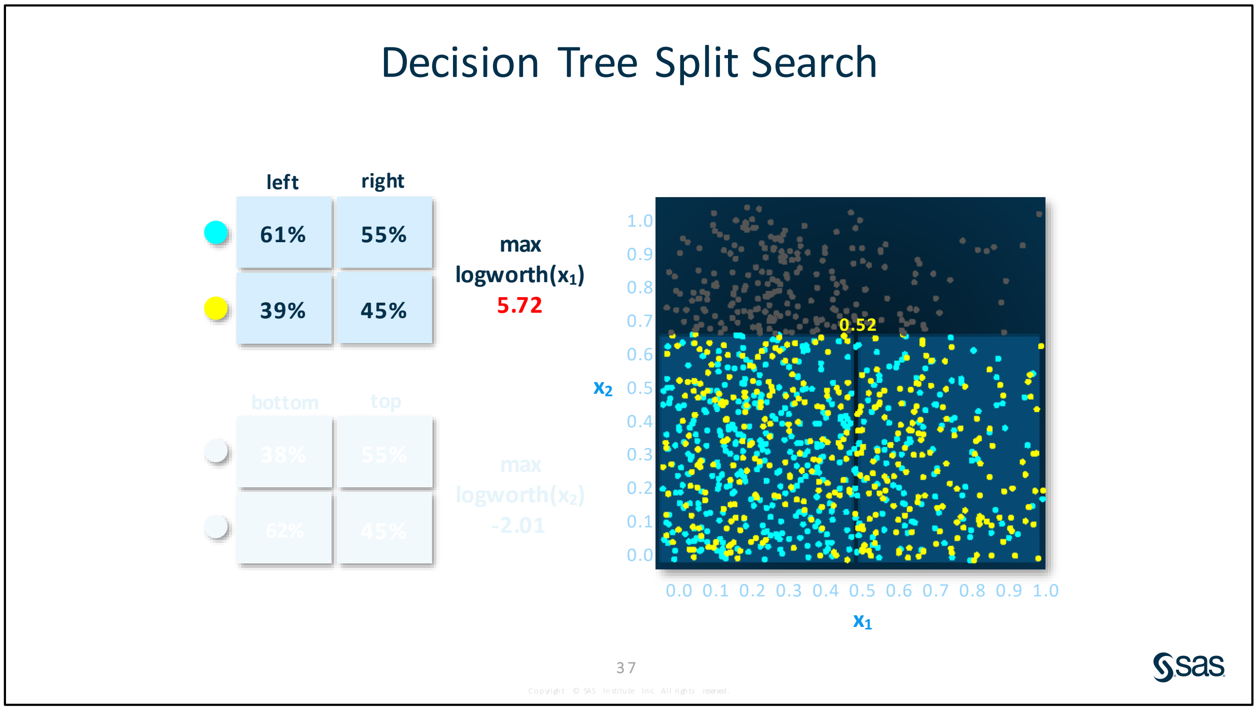
	left	right
●	61%	55%
●	39%	45%

max logworth(x_1) 5.72

	bottom	top
●	38%	55%
●	62%	45%

max logworth(x_2) -2.01

Compare partition logworth ratings.



Decision Tree Split Search

	left	right
●	61%	55%
●	39%	45%

max logworth(x_1) 5.72

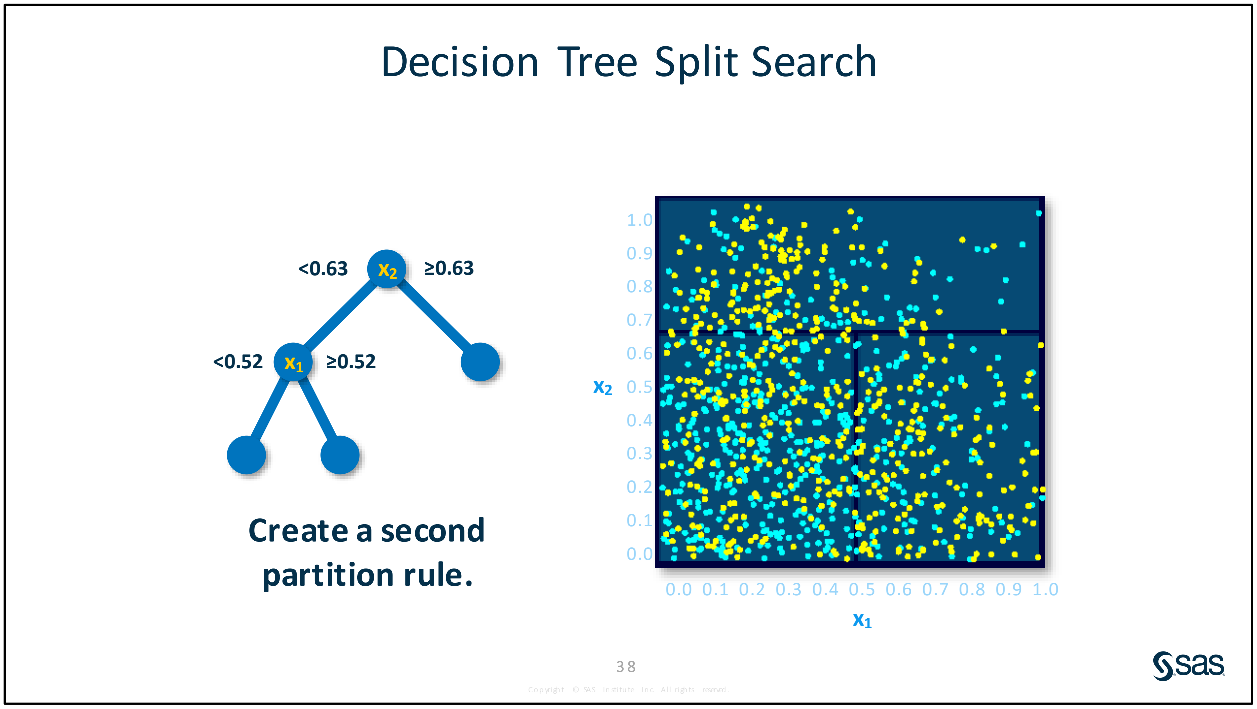
	bottom	top
○	38%	55%
○	62%	45%

max logworth(x_2) -2.01

The split search continues within each leaf. Logworths are compared as before.

Because the significance of secondary and subsequent splits depends on the significance of the previous splits, the algorithm again faces a multiple comparison problem. To compensate for this problem, the algorithm increases the threshold by an amount related to the number of splits above the current split. For binary splits, the threshold is increased by $log_{10}(2)\ d \approx 0.3 \cdot d$, where d is the depth of the split on the decision tree.

Note: By increasing the threshold for each depth (or equivalently decreasing the logworths), the tree algorithm makes it increasingly easy for an input's splits to be excluded from consideration.

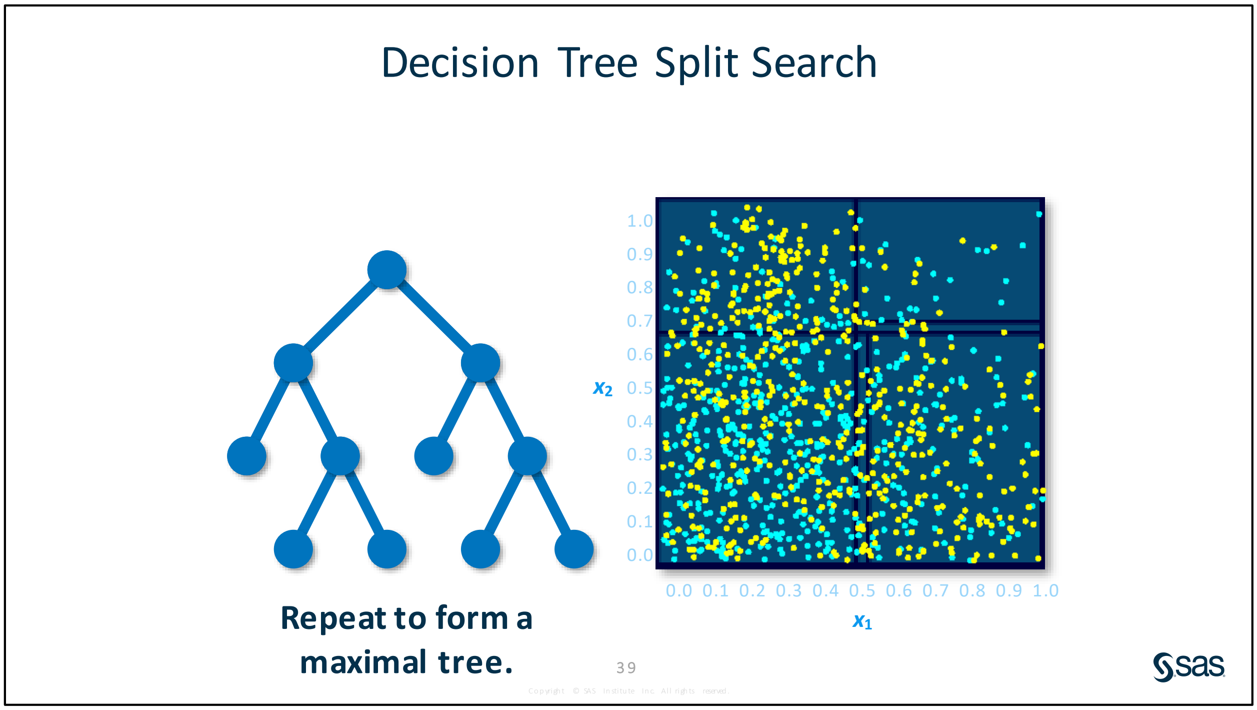


Decision Tree Split Search

Create a second partition rule.

38

The data are partitioned according to the best split, which creates a second partition rule. The process repeats in each leaf until there are no more splits whose adjusted logworth exceeds the depth-adjusted thresholds. This process completes the split-search portion of the tree algorithm.

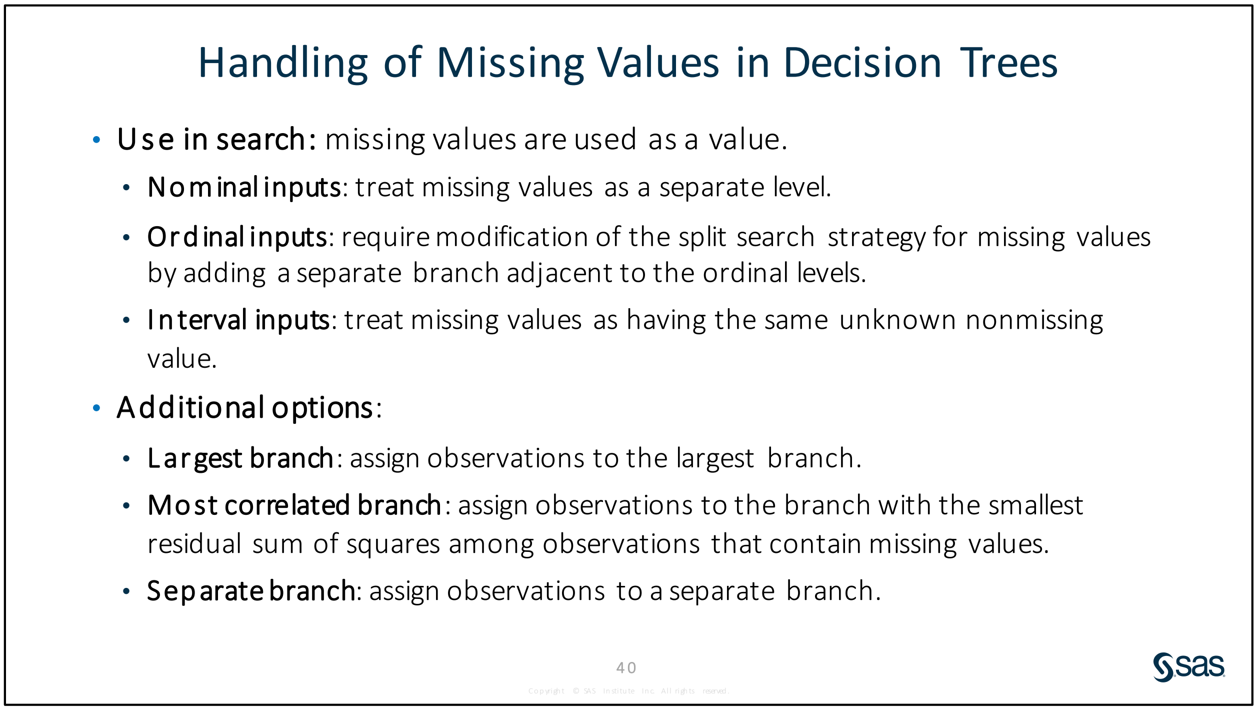


Decision Tree Split Search

Repeat to form a maximal tree.

39

The resulting partition of the input space is known as the *maximal tree*. Development of the maximal tree is based exclusively on statistical measures of split worth on the training data. It is likely that the maximal tree fails to generalize well on an independent set of validation data. The maximal tree is the starting place for how complexity of the model will be optimized. Optimizing the complexity of a tree is done through pruning, and this is covered in the next section.



Handling of Missing Values in Decision Trees

- **Use in search:** missing values are used as a value.
  - **Nominal inputs**: treat missing values as a separate level.
  - **Ordinal inputs**: require modification of the split search strategy for missing values by adding a separate branch adjacent to the ordinal levels.
  - **Interval inputs**: treat missing values as having the same unknown nonmissing value.
- **Additional options**:
  - **Largest branch**: assign observations to the largest branch.
  - **Most correlated branch**: assign observations to the branch with the smallest residual sum of squares among observations that contain missing values.
  - **Separate branch**: assign observations to a separate branch.

40

One of the key benefits of recursive partitioning is the treatment of missing input data. Parametric regression models require complete cases. One missing value on one input variable eliminates that case from analysis. Imputation methods are often used before model fitting to fill in the missing values.

If the value of the target variable is missing, the observation is excluded from training and evaluating the decision tree model.

Decision trees can use missing values in the calculation of the worth of a splitting rule. This consequently produces a splitting rule that assigns the missing values to the branch that maximizes the worth of the split. This is a desirable option when the existence of a missing value is predictive of a target value. Decision trees can use missing values in the split search as a new category or as an unknown numeric nonmissing value.

It treats missing input values as a separate level of the input variable. A nominal input with *L* levels and a missing value can be treated as an *L* + 1 level input. If a new case has a missing value on a splitting variable, then the case is sent to whatever branch contains the missing values.

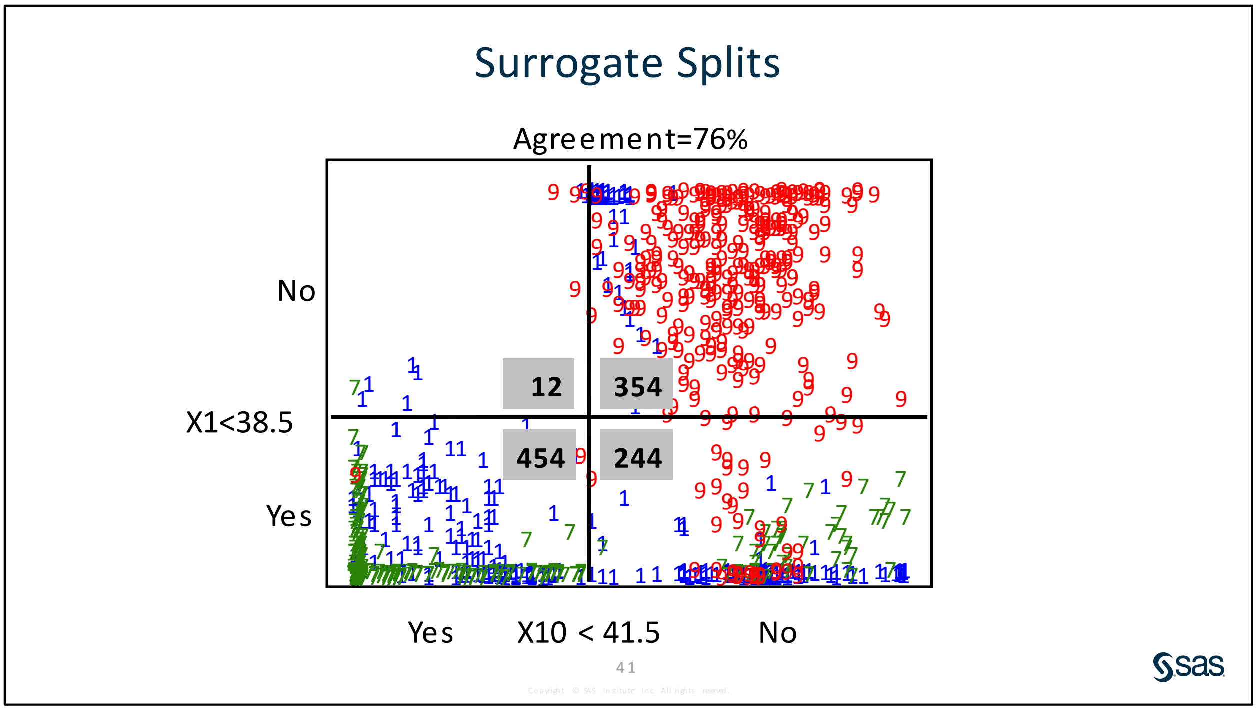
For splits on a categorical variable, this amounts to treating a missing value as a separate category. For numerical variables, it amounts to treating missing values as having the same unknown nonmissing value.

One advantage of using missing data during the search is that the worth of the split is computed with the same number of observations for each input. Another advantage is that an association of the missing values with the target values can contribute to the predictive ability of the split.

The search for a split on an input uses observations whose values are missing on the input. All such observations are assigned to the same branch. The branch might or might not contain other observations. The resulting branch maximizes the worth of the split.

Another option is to not use missing values in the split search. In this case, Decision Trees assigns the observations that contain missing values to a particular branch according to some criteria.

- **Largest branch**: assign observations to the largest branch.
- **Most correlated branch**: assign observations to the branch with the smallest residual sum of squares among observations that contain missing values.
- **Separate branch**: assign observations to a separate branch.



Surrogate Splits

When a split is applied to an observation in which the required input value is missing, surrogate splitting rules can be considered before assigning the observation to the branch for missing values.

A surrogate splitting rule is a backup to the main splitting rule. For example, the main splitting rule might use **COUNTY** as input, and the surrogate might use **REGION**. If the **COUNTY** is unknown and the **REGION** is known, the surrogate is used.

If several surrogate rules exist, each surrogate is considered in sequence until one can be applied to the observation. If none can be applied, the main rule assigns the observation to the branch that is designated for missing values.

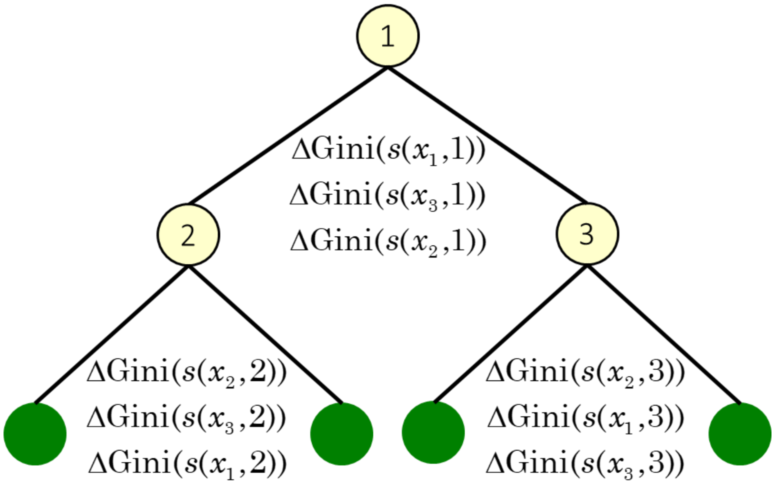
The surrogates are considered in the order of their agreement with the main splitting rule. The agreement is measured as the proportion of training observations that the surrogate rule and the main rule assign to the same branch. The measure excludes the observations to which the main rule cannot be applied. Among the remaining observations, those on which the surrogate rule cannot be applied count as observations that are not assigned to the same branch. Thus, an observation that has used a missing value on the input in the surrogate rule but not the input in the primary rule counts against the surrogate.

The Number of Surrogate Rules property determines the number of surrogates that are sought. A surrogate is discarded if its agreement is less than or equal to the largest proportion of observations in any branch. As a consequence, a node might have fewer surrogates specified than the number in the Number of Surrogate Rules property.

Surrogate splits can be used to handle missing values (BFOS 1984). A surrogate split is a partition using a different input that mimics the selected split. A perfect surrogate maps all the cases that are in the same node of the primary split to the same node of the surrogate split. The agreement between two splits can be measured as the proportion of cases that are sent to the same branch. The split with the greatest agreement is taken as the best surrogate.

When surrogate rules are requested, if a new case has a missing value on the splitting variable, then the best surrogate is used to classify the case. If the surrogate variable is missing as well, then the second-best surrogate is used. If the new case has a missing value on all the surrogates, it is sent to the branch that contains the missing values of the training data.

Details: Variable Importance Based on Gini Reduction


1
2
3
$\Delta\text{Gini}(s(x_1,1))$
$\Delta\text{Gini}(s(x_3,1))$
$\Delta\text{Gini}(s(x_2,1))$
$\Delta\text{Gini}(s(x_2,2))$
$\Delta\text{Gini}(s(x_3,2))$
$\Delta\text{Gini}(s(x_1,2))$
$\Delta\text{Gini}(s(x_2,3))$
$\Delta\text{Gini}(s(x_1,3))$
$\Delta\text{Gini}(s(x_3,3))$

BFOS (1984) devised a measure of variable importance for trees. It can be particularly useful for tree interpretation.

Let $s(x_j,t)$ be a surrogate split (including the primary split) at the *t*th internal node using the *j*th input. Importance is a weighted average of the reduction in impurity for the surrogate splits using the *j*th input across all the internal nodes in the tree. The weights are the node sizes.

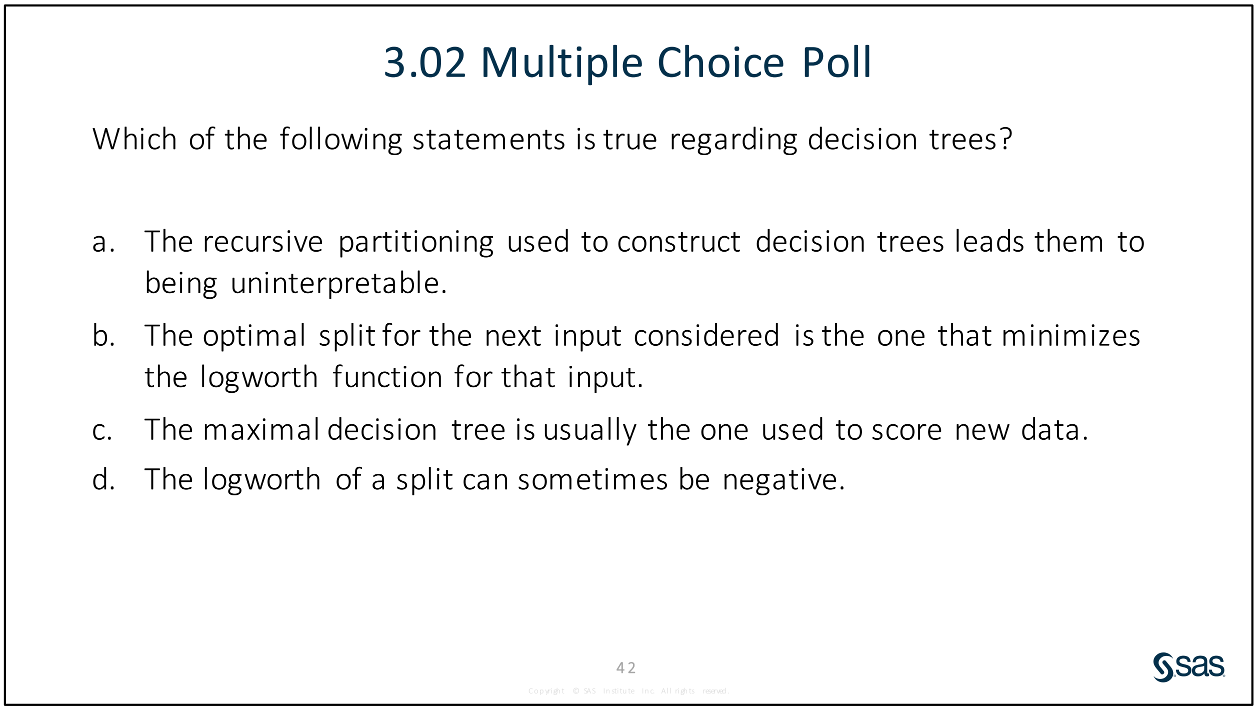
$$\text{Importance}(x_j) = \sum_{t=1}^{T} \frac{n_t}{n} \Delta(i)(s(x_j),t),$$

where $\Delta(i)$ represents impurity reduction. For interval targets, variance reduction is used. For categorical targets, variance reduction reduces to Gini reduction.

In the Decision Tree task, variable importance is calculated similarly to BFOS (1984), although it takes the square root. Further, the Decision Tree node incorporates the agreement between the surrogate split and the primary split in the calculation. The variable importance measure is scaled to be between 0 and 1 by dividing by the maximum importance. Thus, larger values indicate greater importance. Variables that do not appear in any primary or saved surrogate splits have 0 importance.

One major difference between variable importance in the Decision Tree task and in BFOS (1984) is that, by default, surrogates are not saved. Therefore, they are not included in the calculation. This practice disregards a fundamental purpose of variable importance: unmasking inputs that have splits that are correlated with primary splits.

If two variables are highly correlated and they are both used in primary splitting rules, they dilute each other's importance. Requesting surrogates remedies this, and also remedies the situation where one of the two variables happens not to appear in any primary splitting rule.



3.02 Multiple Choice Poll

Which of the following statements is true regarding decision trees?

a. The recursive partitioning used to construct decision trees leads them to being uninterpretable.

b. The optimal split for the next input considered is the one that minimizes the logworth function for that input.

c. The maximal decision tree is usually the one used to score new data.

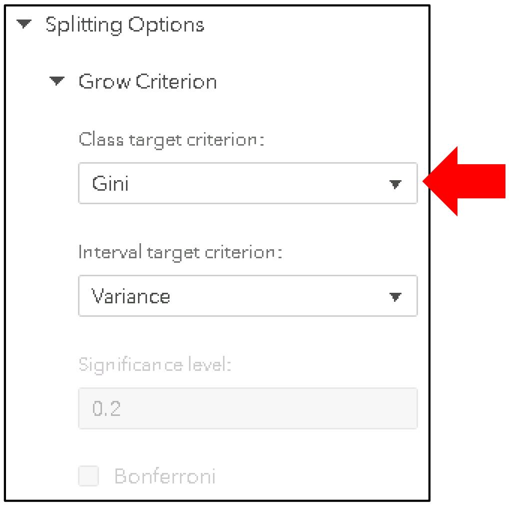
d. The logworth of a split can sometimes be negative.

Improving a Decision Tree Model by Changing the Recursive Partitioning Parameters

In this demonstration, you change more settings of the Decision Tree node in the Starter Template pipeline. You modify the recursive partitioning parameters and compare this model performance to the models built earlier in the course.

Recall that the previous model, based on changing the tree structure parameters, achieved an average square error of 0.0801 on the VALIDATE partition. We will try to improve the model's performance by modifying some of the settings of the Decision Tree model.

1. Under the **Grow Criterion** properties, change **Class target criterion** from **Information gain ratio** to **Gini**.



▼ Splitting Options

▼ Grow Criterion

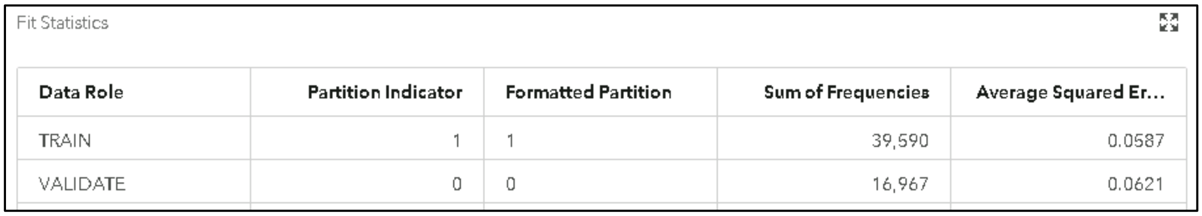
Class target criterion: Gini ▾

Interval target criterion: Variance ▾

Significance level: 0.2

□ Bonferroni

2. Run the Decision Tree node.
3. Open the results for the Decision Tree node.
4. Click the **Assessment** tab.



Fit Statistics

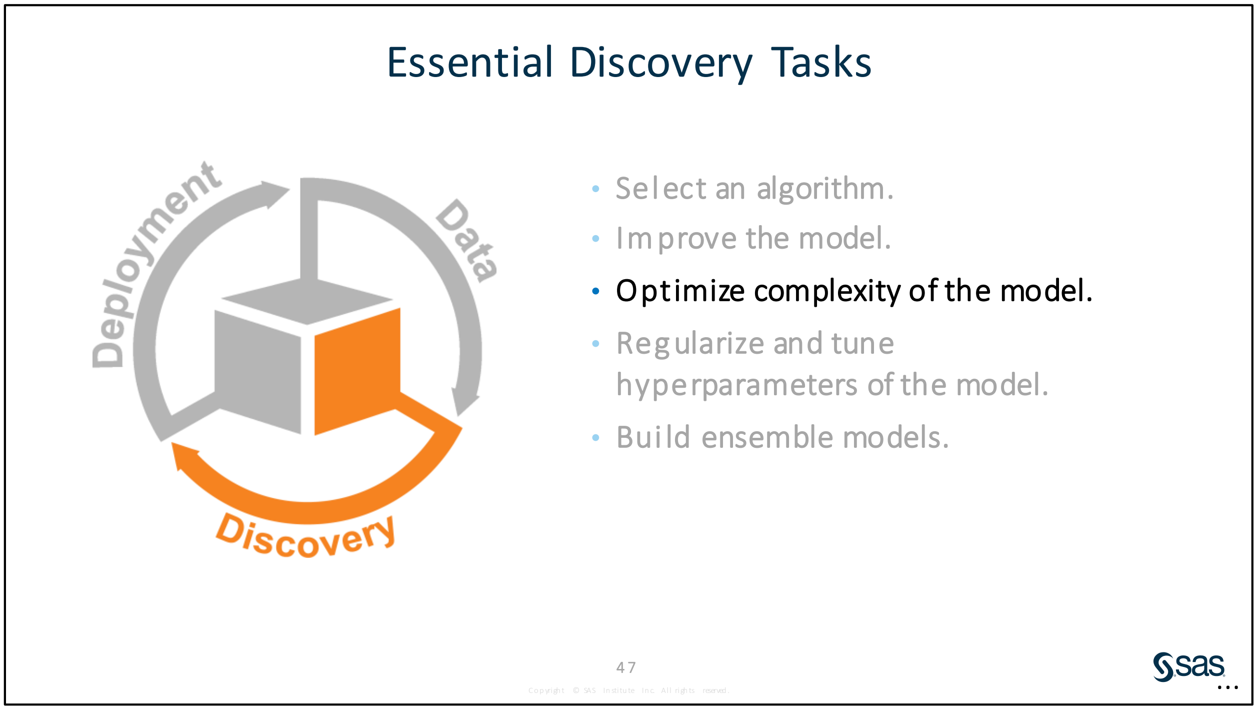
Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.0587
VALIDATE	0	0	16,967	0.0621

The average square error for the tuned Decision Tree model is 0.0621 on the VALIDATE partition. This fit statistic is approximately 22% better than the previous model by changing only the recursive partitioning parameters.

5. Close the Results window.

End of Demonstration

3.4 Pruning

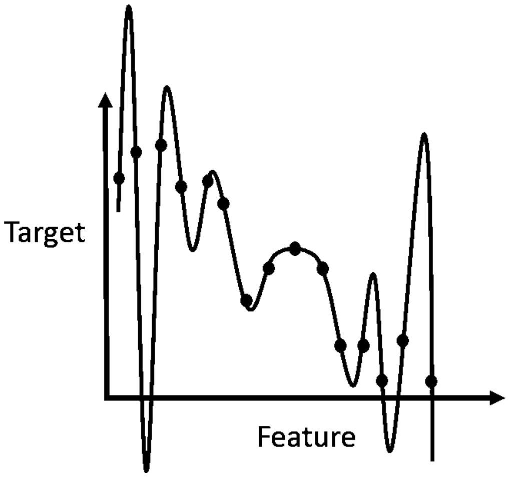


Essential Discovery Tasks

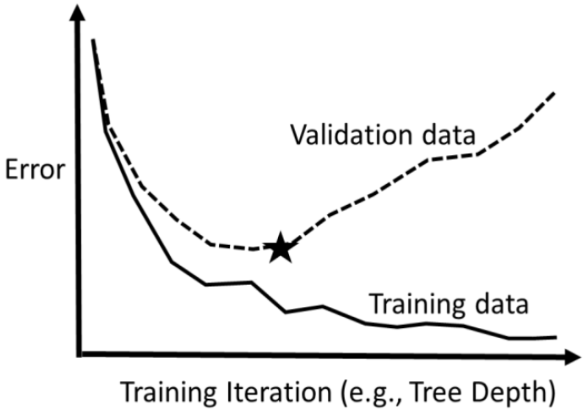
- Select an algorithm.
- Improve the model.
- **Optimize complexity of the model.**
- Regularize and tune hyperparameters of the model.
- Build ensemble models.

47

Recall that the goal is to build models that can be used to score future observations to enable you to make business decisions, such as to churn or not churn, flag fraudulent activity, predict potential revenue, and so on. Machine learning algorithms are very effective at learning a mapping between the features and known target values in your existing data; if left unattended, they can often create a 100% accurate mapping, as shown below.



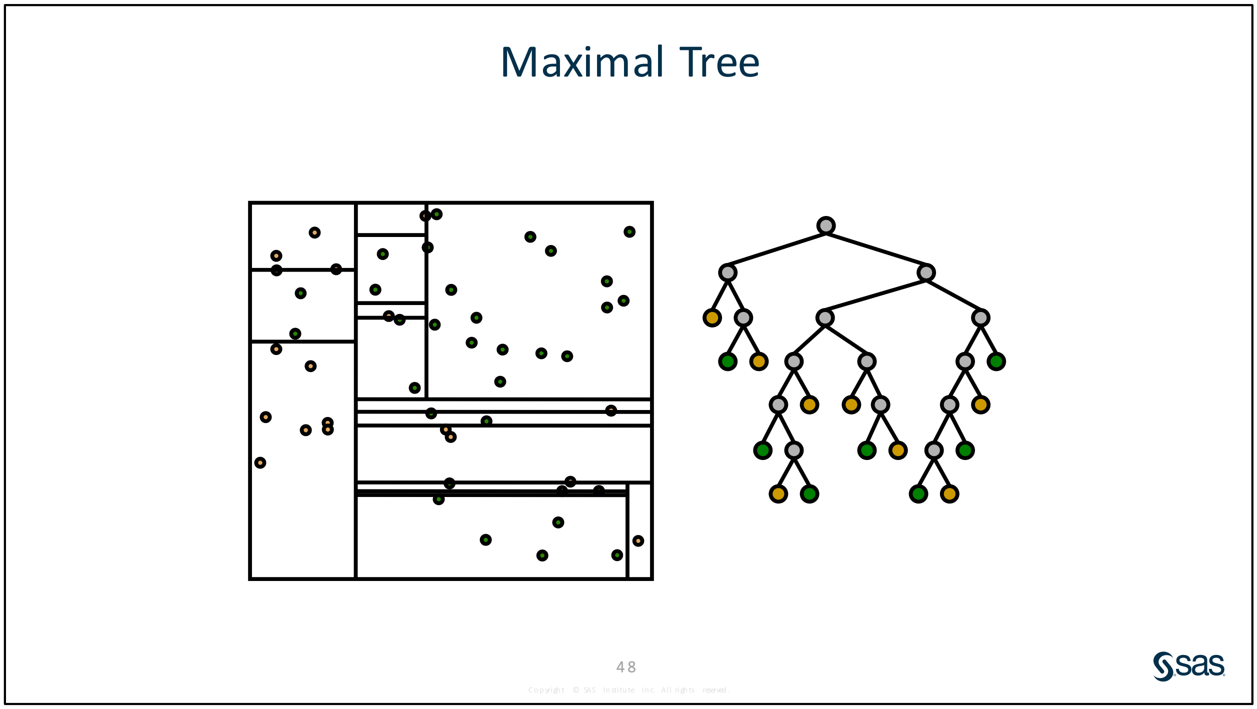
Example of Overfitting



Training and Validation Error Compromise

Evidently, a model that is complex enough to perfectly fit the existing data will not generalize well when used to score new observations. It might provide accurate answers for some cases by chance, but in general it does not represent the trend of the data. This is referred to as overfitting. A decision tree is a prime example of an algorithm that can easily overfit the data. If the tree can continue to split the data all the way down to each observation being in its own leaf, it will be 100% accurate for every observation in the training data. But after a certain depth, the tree is not providing any information that can be applied in general.

Honest assessment, which is highly related to the bias-variance tradeoff, involves calculating error metrics from scoring the model on data that were not used in any way during the training process. The distinctions between the validation data and the test data, and to incorporate them as part of your model training, assessment, and selection process has briefly been discussed in Chapter 1.



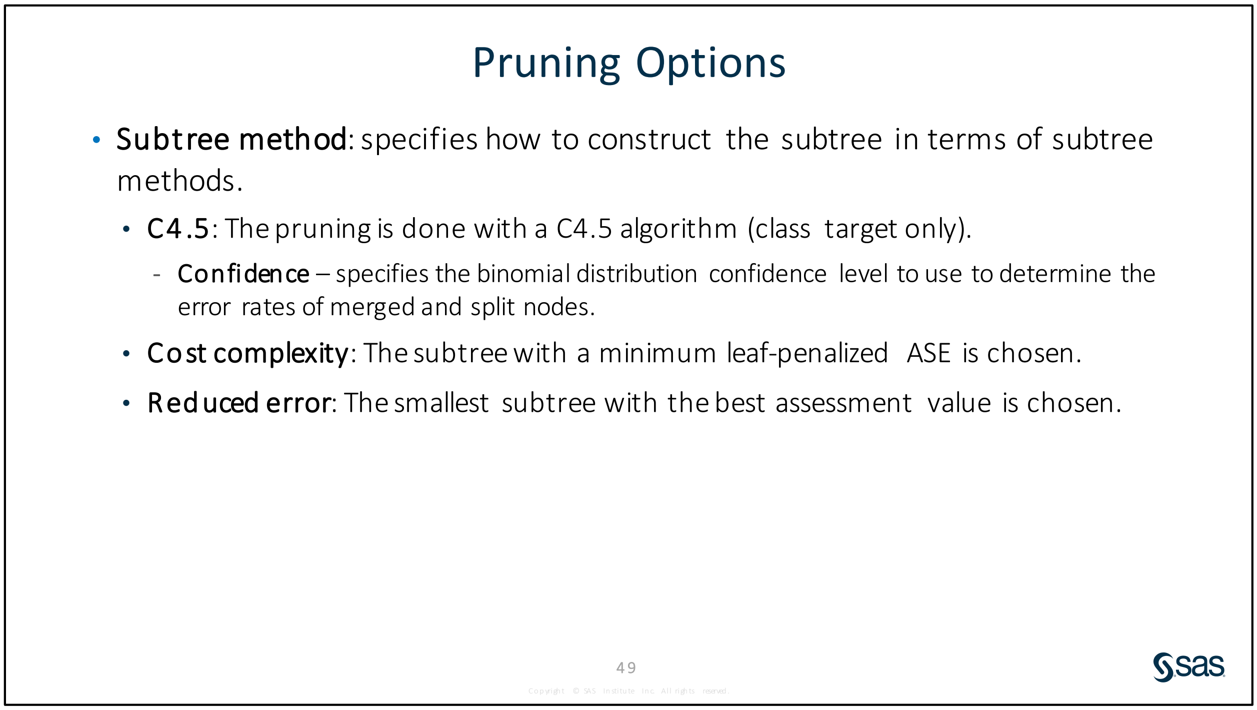
Maximal Tree

48

A large decision tree can be grown until every node is as pure as possible. If at least two observations have the same values on the input variables but different target values, it is not possible to achieve perfect purity. The tree with the greatest possible purity on the training data is the *maximal classification* tree.

The maximal tree is the result of overfitting. It adapts to both the systematic variation of the target (signal) and the random variation (noise). It usually does not generalize well on new (noisy) data.

A small tree with only a few branches might underfit the data. It might fail to adapt sufficiently to the signal. This usually results in poor generalization.



Pruning Options

- **Subtree method**: specifies how to construct the subtree in terms of subtree methods.
  - **C4.5**: The pruning is done with a C4.5 algorithm (class target only).
    - **Confidence** – specifies the binomial distribution confidence level to use to determine the error rates of merged and split nodes.
  - **Cost complexity**: The subtree with a minimum leaf-penalized ASE is chosen.
  - **Reduced error**: The smallest subtree with the best assessment value is chosen.

49

Tree complexity is a function of the number of leaves, the number of splits, and the depth of the tree. Determining complexity is crucial with flexible models like decision trees. A well-fit tree has low bias (adapts to the signal) and low variance (does not adapt to the noise). The determination of model complexity usually involves a tradeoff between bias and variance. An underfit tree that is not sufficiently complex has high bias and low variance. In contrast, an overfit tree has low bias and high variance.

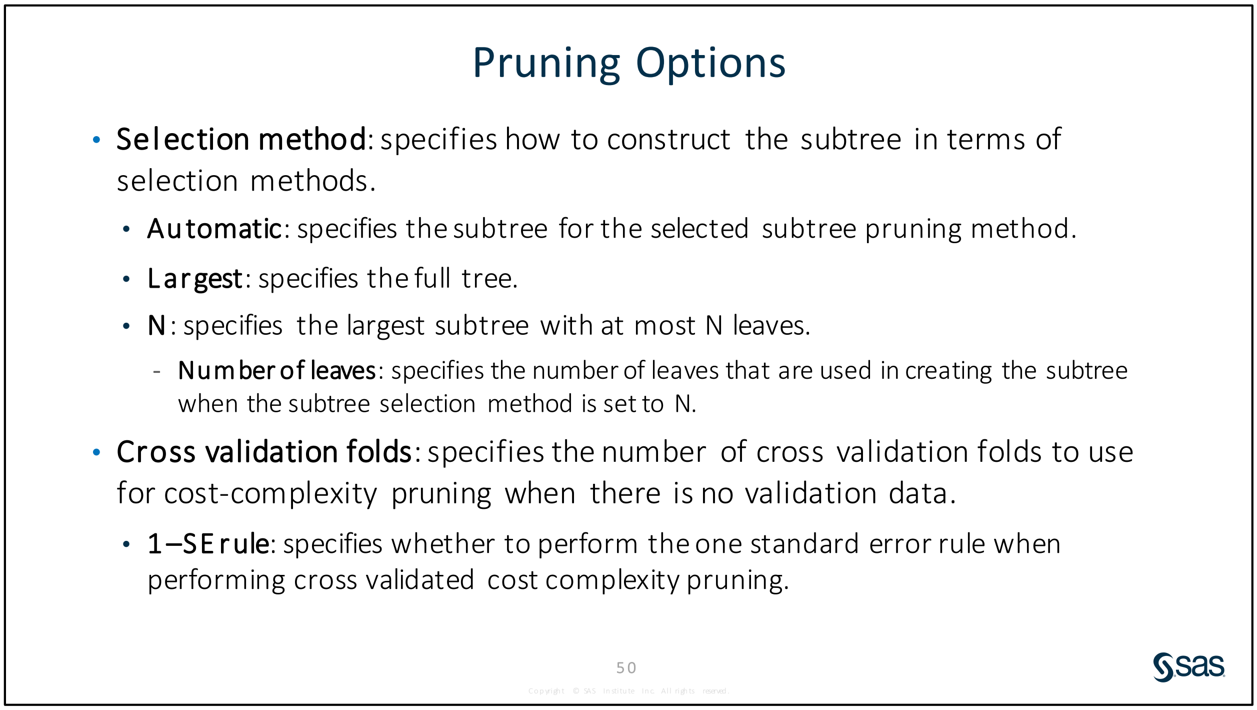
The maximal tree represents the most complicated model that you are willing to construct from a set of training data. To avoid potential overfitting, many predictive modeling procedures offer some mechanism for adjusting model complexity. For decision trees, this process is known as *pruning*.

The subtree method specifies how to construct the subtree in terms of subtree methods. Possible values include the following:

- **C4.5**: The pruning is done with a C4.5 algorithm.
- **Cost complexity**: The subtree with a minimum leaf-penalized ASE is chosen.
- **Reduced error**: The smallest subtree with the best assessment value is chosen.

The C4.5 algorithm is available only for class targets.

With reduced error pruning, the assessment measure for class targets is misclassification rate, and the assessment measure for interval targets is ASE.



Pruning Options

- **Selection method**: specifies how to construct the subtree in terms of selection methods.
  - **Automatic**: specifies the subtree for the selected subtree pruning method.
  - **Largest**: specifies the full tree.
  - **N**: specifies the largest subtree with at most N leaves.
    - **Number of leaves**: specifies the number of leaves that are used in creating the subtree when the subtree selection method is set to N.
- **Cross validation folds**: specifies the number of cross validation folds to use for cost-complexity pruning when there is no validation data.
  - **1–SE rule**: specifies whether to perform the one standard error rule when performing cross validated cost complexity pruning.

50

The selection method specifies how to construct the subtree in terms of selection methods. Possible values include the following:

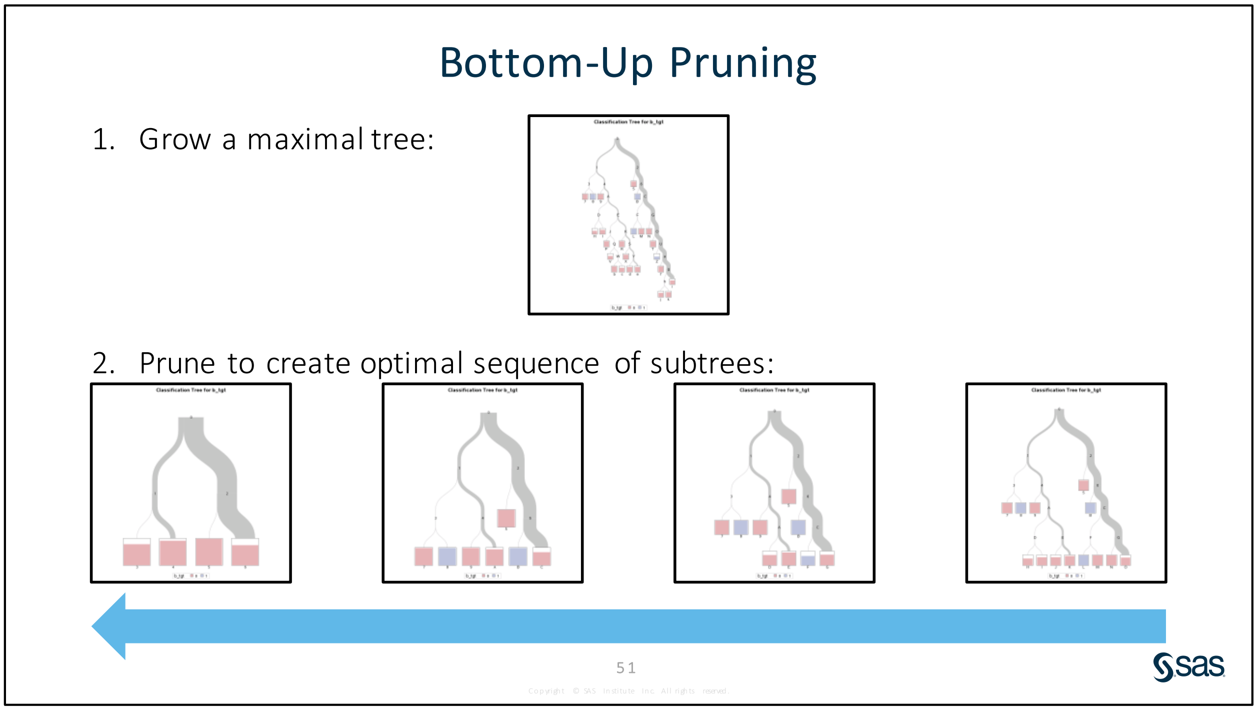
- **Automatic**: specifies the appropriate subtree for the specified subtree pruning method.
- **Largest**: specifies the full tree.
- **N**: specifies the largest subtree with at most N leaves.

The Number of leaves property specifies the number of leaves that are used in creating the subtree when the subtree selection method is set to N.

The Confidence property specifies the binomial distribution confidence level to use to determine the error rates of merged and split nodes. The default value is 0.25. This option is available only when C4.5 is the pruning method.

The Cross validation folds property specifies the number of cross validation folds to use for cost-complexity pruning when there is no validation data. Possible values range from 2 to 20.

The 1–SE rule property specifies whether to perform the one standard error rule when performing cross validated cost complexity pruning.



Bottom-Up Pruning

1. Grow a maximal tree:

2. Prune to create optimal sequence of subtrees:

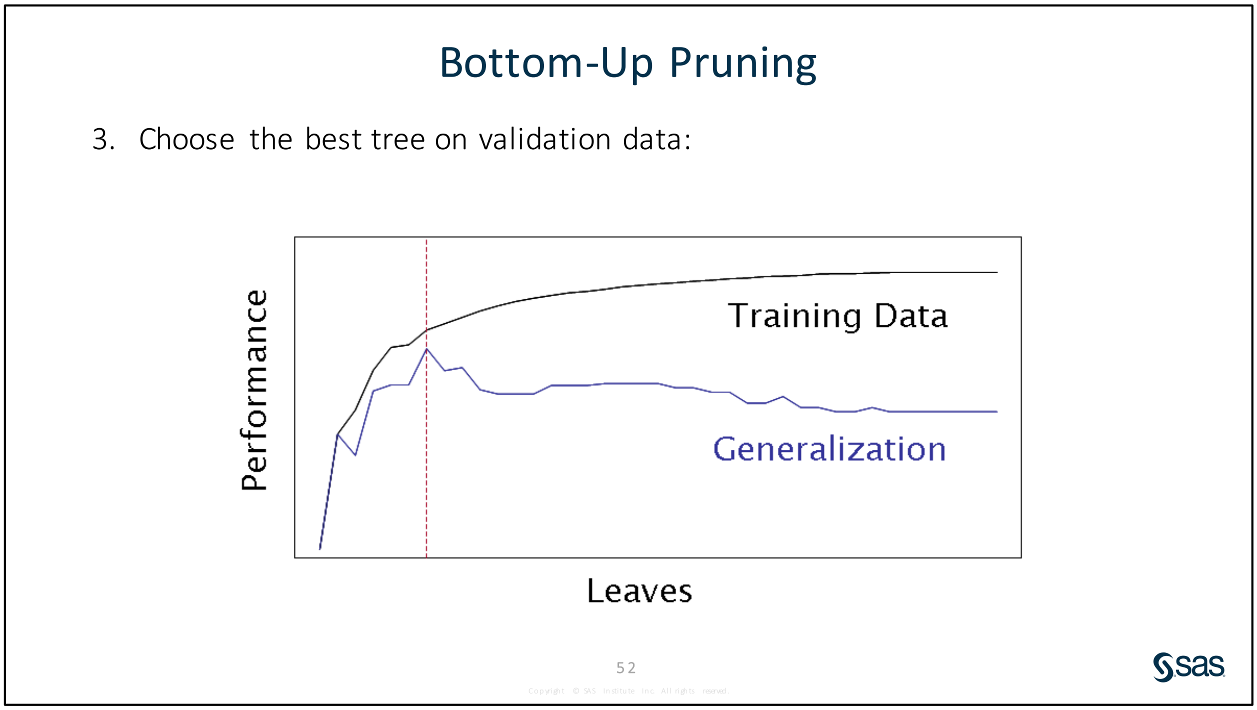
Classification Tree for b_tgt
Classification Tree for b_tgt
Classification Tree for b_tgt
Classification Tree for b_tgt
Classification Tree for b_tgt

In bottom-up (post) pruning, a large tree is grown and then branches are lopped off in a backward fashion using some model selection criterion. The bottom-up strategy of intentionally creating more nodes than will be used is also called *retrospective pruning* and originated with cost-complexity pruning (BFOS 1984).

For any subtree, T, in a tree grown from 1 to *n* leaves, define its complexity or size (number of leaves) as L, and define R(L) as the validation set misclassification cost. Other assessment measures can also be used.

In SAS Viya, the pruning process starts with the maximal tree T_{max} with L leaves. The maximal tree is denoted as T_L. Construct a series of smaller and smaller trees T_L, T_{L-1}, T_{L-2}, …, T_1, such that the following holds: For every value of H*i*, where $1 \leq Hi \leq L$, consider the class T_{Hi} of all subtrees of size H_i. Select the subtree in the series that minimizes R(T_{Hi}).

Note: The trees in the series of subtrees are not necessarily nested.



Bottom-Up Pruning

3. Choose the best tree on validation data:

The subtree with the best performance on validation data is selected.

Top-down pruning is usually faster but is considered less effective than bottom-up pruning. Breiman and Friedman, in their criticism of the FACT tree algorithm (Loh and Vanichsetakul 1988), discussed their experiments with stopping rules as part of the development of the CART methodology:

> "Each stopping rule was tested on hundreds of simulated data sets with different structures. Each new stopping rule failed on some data set. It was not until a very large tree was built and then pruned, using cross validation to govern the degree of pruning, that we observed something that worked consistently."

Bottom-up pruning has two requirements:

- A method for honestly measuring performance.

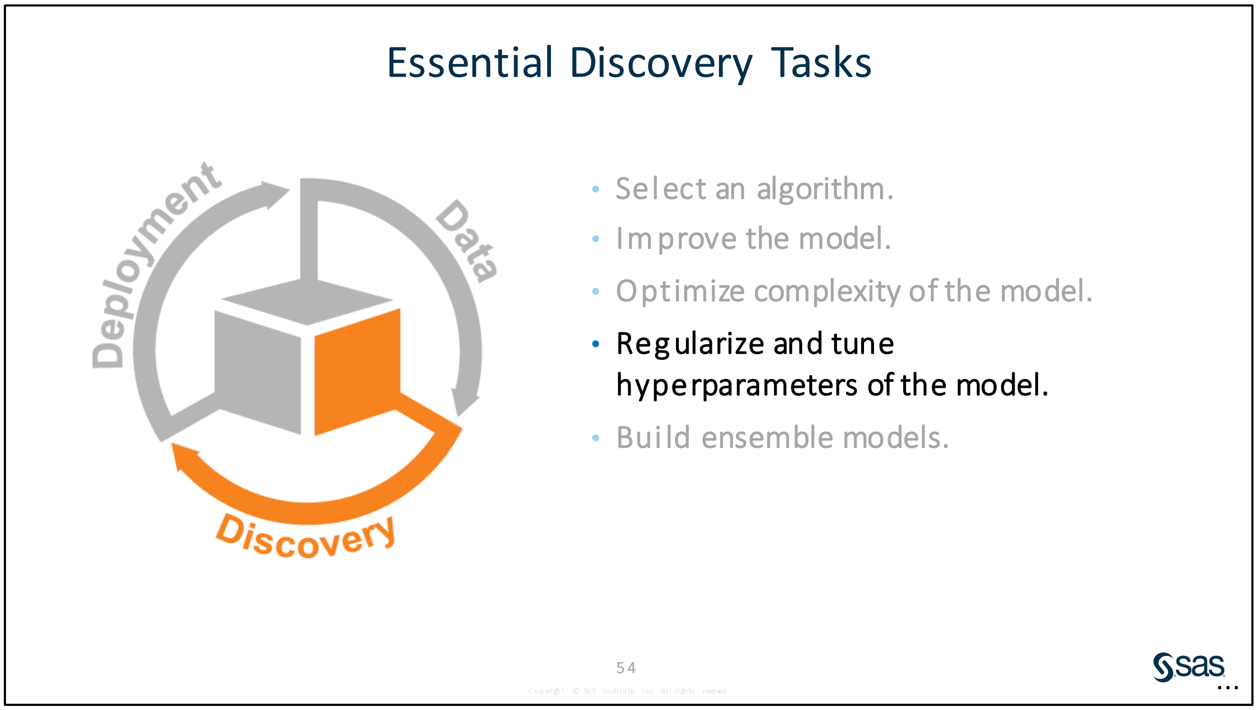
  The simplest remedy is to split the data into training and validation sets. The validation data are used for model comparison. Data splitting is inefficient when the data are small. Removing data from the training set can degrade the fit. Furthermore, evaluating performance on a single validation set can give imprecise results.

  A more efficient remedy—but more computationally expensive—is *k*-fold cross validation. In *k*-fold cross validation, performance measures are averaged over *k* models. Each model is fit with (*k*−1)/*k* of the data and assessed on the remaining 1/*k* of the data. The average over the *k* holdout data sets is then used to honestly estimate the performance for the model fitted to the full data set. Cross validation is discussed later in this chapter.

- A relevant model selection criterion.

For classification problems, the most appropriate measures of generalization depend on the number of correct and incorrect classifications and their consequences.

For many purposes, including analyses with interval targets, average square error has been found to work very well as a general method for selecting a subtree on the validation data. It is recommended for most practical situations and, in particular, in situations with rare target levels and in which the benefits or costs of correct or incorrect classification are not easy to specify.



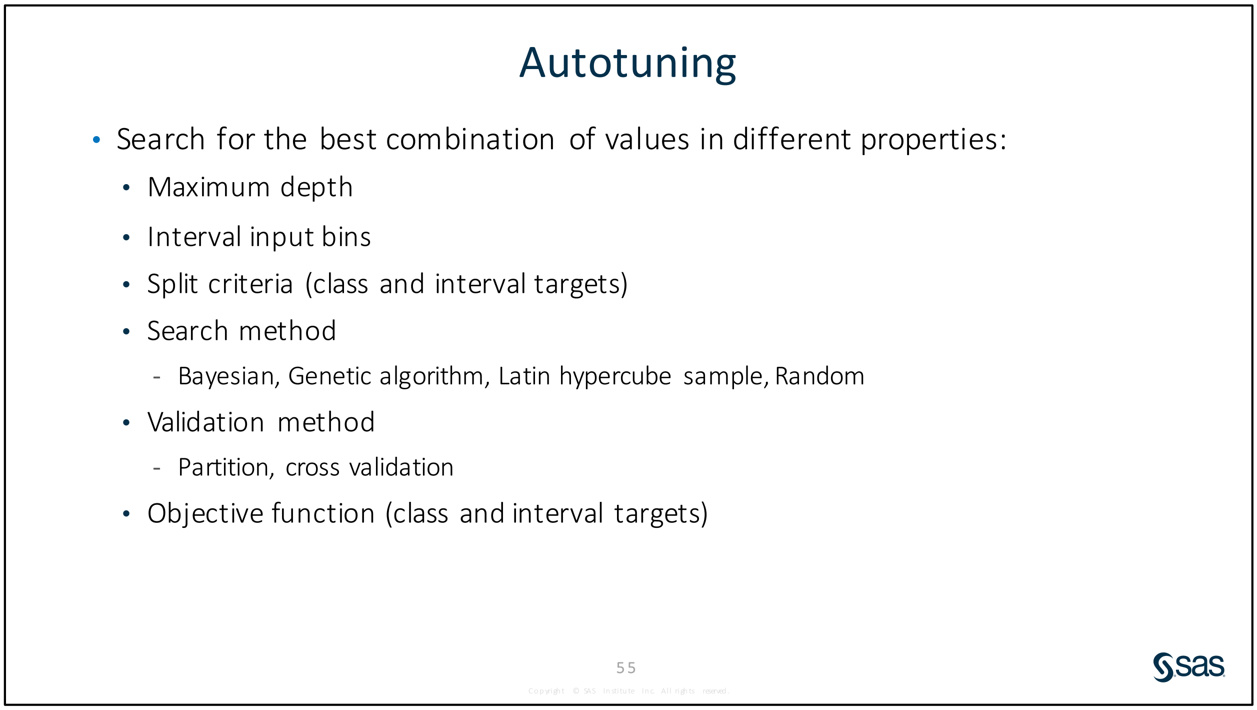
Essential Discovery Tasks

Deployment
Data
Discovery

- Select an algorithm.
- Improve the model.
- Optimize complexity of the model.
- Regularize and tune hyperparameters of the model.
- Build ensemble models.

54

The objective of a machine learning algorithm is to find the model parameters that minimize the loss function over the independent samples. For example, these parameters could be maximum depth or split criteria in a decision tree. As the complexity of your model increases, its predictive abilities often decrease after a certain point due to overfitting and multicollinearity issues. Hence, the resulting models often do not generalize well to new data, and they yield unstable parameter estimates.



Autotuning

- Search for the best combination of values in different properties:
  - Maximum depth
  - Interval input bins
  - Split criteria (class and interval targets)
  - Search method
    - Bayesian, Genetic algorithm, Latin hypercube sample, Random
  - Validation method
    - Partition, cross validation
  - Objective function (class and interval targets)

55

Autotuning searches for the best combination of the decision tree parameters. ***Performing autotuning can substantially increase run time.***

Autotuning runs based on some options, which limit the search of all possible combinations in terms of the decision tree parameters.

Maximum Depth specifies whether to autotune the maximum depth parameter. It ranges from 1 to 150. The default initial value for the maximum depth is 10. The default for the range is from 1 to 19.

Interval input bins specifies whether to autotune the number of interval input bins. It ranges from 2 to 500. The default initial value for the number of bins is 20. The default for the range is from 20 to 200.

Grow Criterion specifies whether to autotune the grow criterion. For class target, the options are Entropy, CHAID, Information gain ratio, Gini, and Chi-square. For interval target, the options are Variance, F test, and CHAID.

Search Options specifies the options for autotuning searching. The following options are available:

- **Genetic algorithm** uses an initial Latin hypercube sample that seeds a genetic algorithm. The genetic algorithm generates a new population of alternative configurations at each iteration.
- **Latin hypercube sample** performs an optimized grid search that is uniform in each tuning parameter, but random in combinations.
- **Random** generates a single sample of purely random configurations.
- **Bayesian** uses priors to seed the iterative optimization.

Number of evaluations per iteration specifies the number of tuning evaluations in one iteration. This option is available only if the search method is Genetic algorithm or Bayesian. The default value is 10. It ranges from 2 to 2,147,483,647.

Maximum number of evaluations specifies the maximum number of tuning evaluations. This option is available only if the Search method is Genetic algorithm or Bayesian. The default value is 50. It ranges from 3 to 2,147,483,647.

Maximum number of iterations specifies the maximum number of tuning iterations. This option is available only if the search method is Genetic algorithm or Bayesian. The default value is 5. It ranges from 1 to 2,147,483,647.

Sample size specifies the sample size. This option is available only if the search method is Random or Latin hypercube sample. The default value is 50. It ranges from 2 to 2,147,483,647.

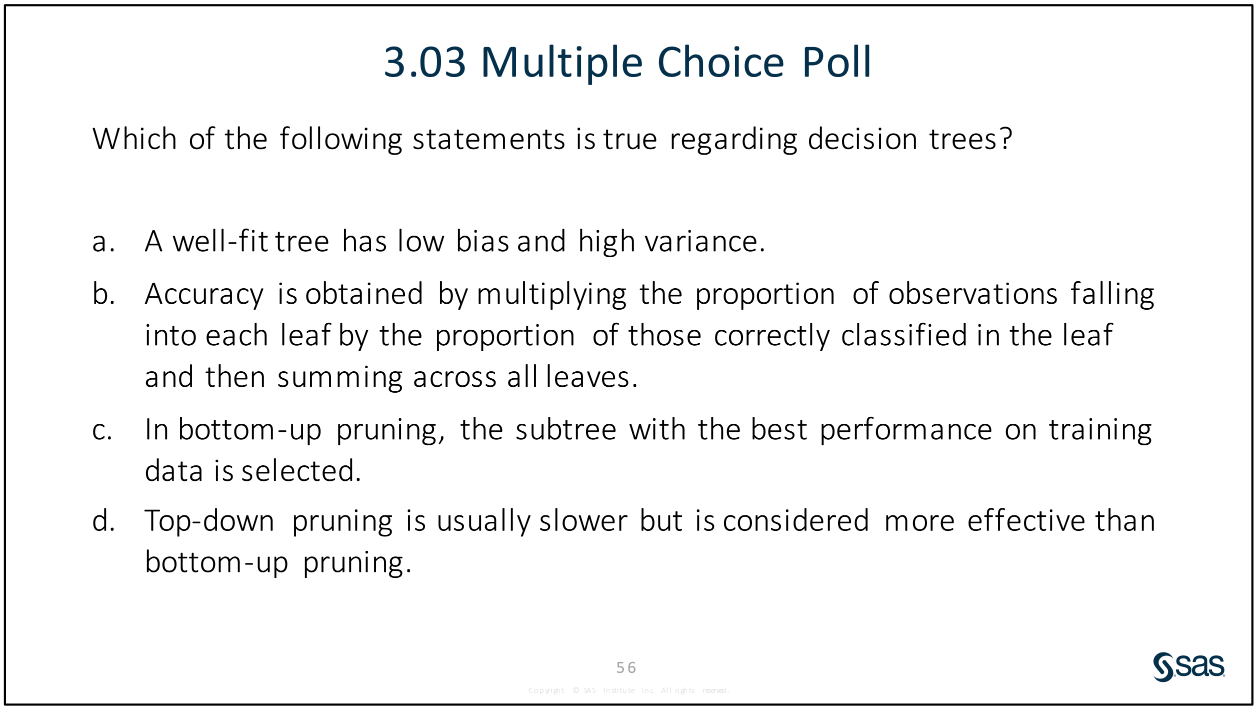
There are some general options associated with the autotuning search.

Validation method specifies the validation method for finding the objective value. If your data is partitioned, then that partition is used. Validation method, Validation data proportion, and Cross validation number of folds are all ignored.

- **Partition** specifies using the partition validation method. With partition, you specify proportions to use for randomly assigning observations to each role.
  - **Validation data proportion** specifies the proportion of data to be used for the partition validation method. The default value is 0.3.
- **K-fold cross validation** specifies using the cross validation method. In cross validation, each model evaluation requires k training executions (on k-1 data folds) and k scoring executions (on one holdout fold). This increases the evaluation time by approximately a factor of k.
  - **Cross validation number of folds** specifies the number of partition folds in the cross validation process (the k defined above). Possible values range from 2 to 20. The default value is 5.

Nominal target objective function specifies the objective function to optimize for tuning parameters for a nominal target. Possible values are average square error, area under the curve, F1 score, F0.5 score, gamma, Gini coefficient, Kolmogorov-Smirnov statistic, multi-class log loss, misclassification rate, root average squared error, and Tau. The default value is misclassification rate.

Interval target objective function specifies the objective function to optimize for tuning parameters for an interval target. Possible values are average squared error, mean absolute error, mean square logarithmic error, root average squared error, root mean absolute error, and root mean square logarithmic error. The default value is average squared error.



3.03 Multiple Choice Poll

Which of the following statements is true regarding decision trees?

a. A well-fit tree has low bias and high variance.

b. Accuracy is obtained by multiplying the proportion of observations falling into each leaf by the proportion of those correctly classified in the leaf and then summing across all leaves.

c. In bottom-up pruning, the subtree with the best performance on training data is selected.

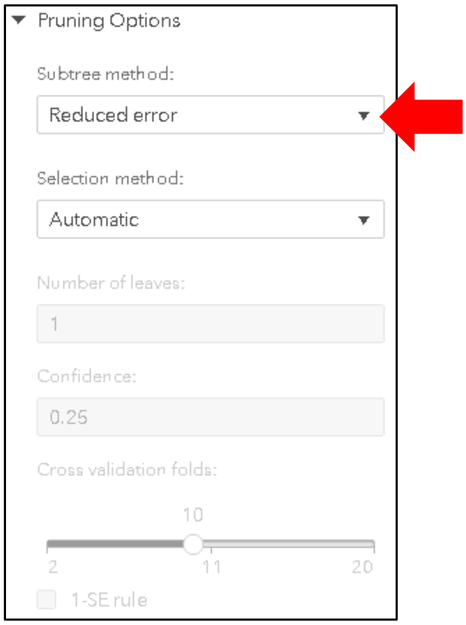
d. Top-down pruning is usually slower but is considered more effective than bottom-up pruning.

56

Improving a Decision Tree Model by Changing the Pruning Parameters

In this demonstration, you change the default settings of the Decision Tree node in the Starter Template pipeline. You modify the pruning parameters and compare this model performance to the model built earlier in the course.

1. To recall, the previous model, based on changings on the tree structure and the recursive partitioning parameters, achieved an average square error of 0.0621 on the Validate partition.
2. Try to improve the model's performance by modifying some of the settings of the Decision Tree model. In the properties pane, expand the properties under **Pruning Options**.
3. Change **Subtree method** from **Cost complexity** to **Reduced error**.



Pruning Options

Subtree method: Reduced error

Selection method: Automatic

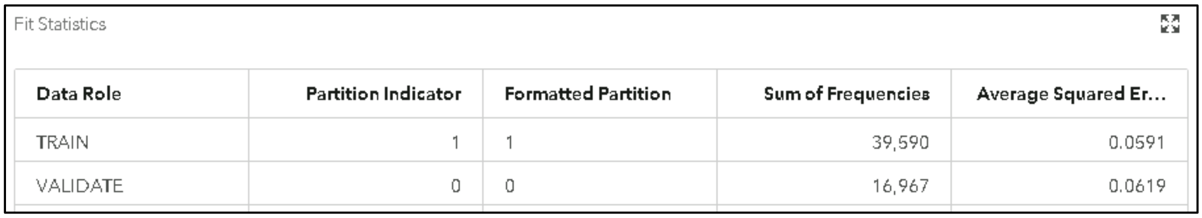
Number of leaves: 1

Confidence: 0.25

Cross validation folds: 10

□ 1-SE rule

4. Run the Decision Tree node.
5. Open the results for the Decision Tree node.
6. Click the **Assessment** tab.

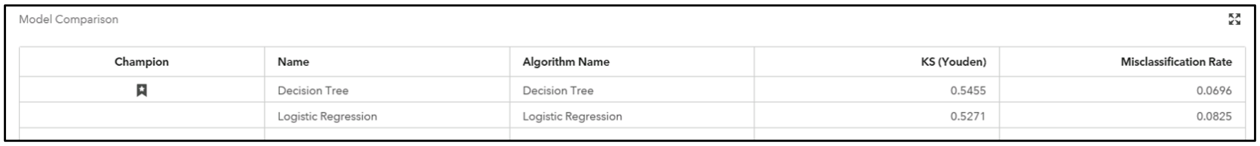


Fit Statistics

Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.0591
VALIDATE	0	0	16,967	0.0619

The average square error for the tuned Decision Tree model is 0.0619 on the VALIDATE partition. This fit statistic is slightly better than the model tuned based on the tree structure and the recursive partitioning parameters, by approximately 1%.

7. Close the Results window.
8. Run the entire pipeline and view the results of the Model Comparison node. The Model Comparison table shows that the Decision Tree model is currently the champion from the Starter Template pipeline. This is based on the default fit statistic KS.



Model Comparison

Champion	Name	Algorithm Name	KS (Youden)	Misclassification Rate
🔖	Decision Tree	Decision Tree	0.5455	0.0696
	Logistic Regression	Logistic Regression	0.5271	0.0825

9. Close the results of the Model Comparison node.

End of Demonstration

Exercises

1. **Building a Decision Tree**

   a. Build a decision tree using the Autotune feature. Add a Decision Tree node to the Starter pipeline, below the Variable Selection node. Use the Autotune feature. Explore the settings that are made available when **Autotune** is selected. Run a few distinct models by changing the range of the parameters available in the Autotune option.

   Note: This exercise might take several minutes to run.

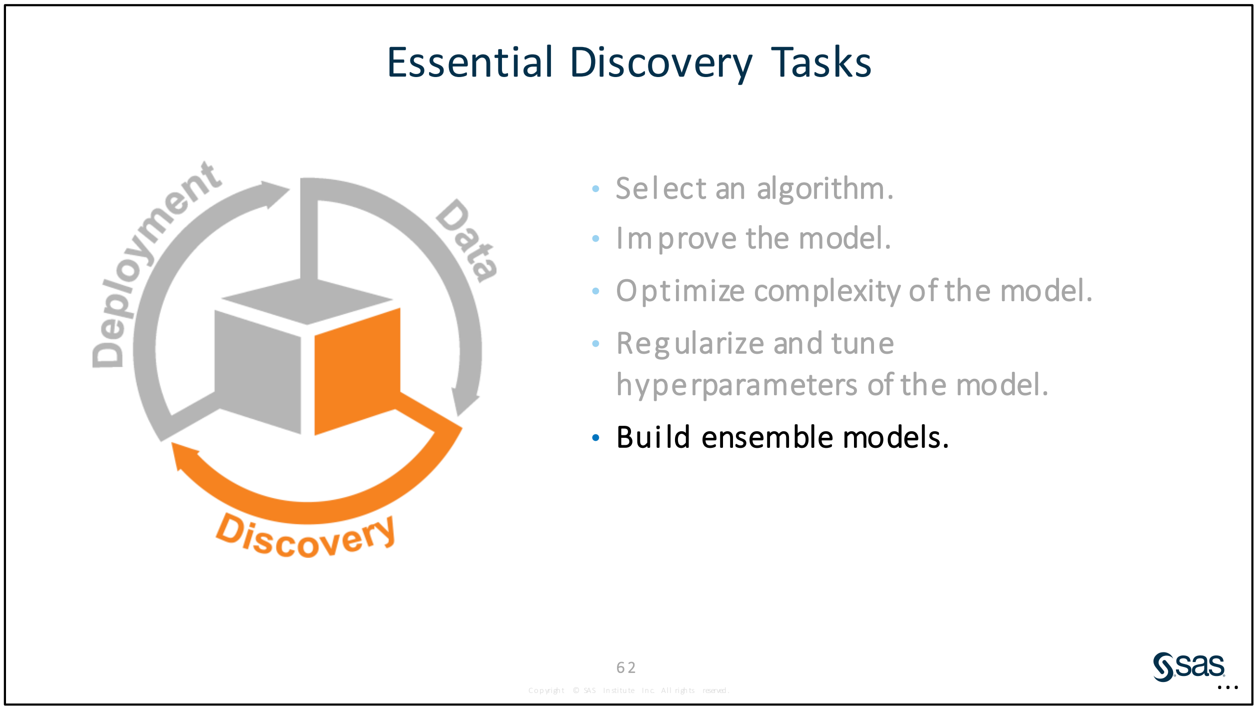
   b. What criteria were selected for the champion model?

   - Split criteria
   - Pruning method
   - Maximum number of branches
   - Maximum tree depth

   c. How does the autotuned decision tree compare to the other models in the pipeline, particularly to the Decision Tree model built during the demonstration? Consider the fit statistic average square error for this comparison.

End of Exercises

3.5 Ensembles of Trees

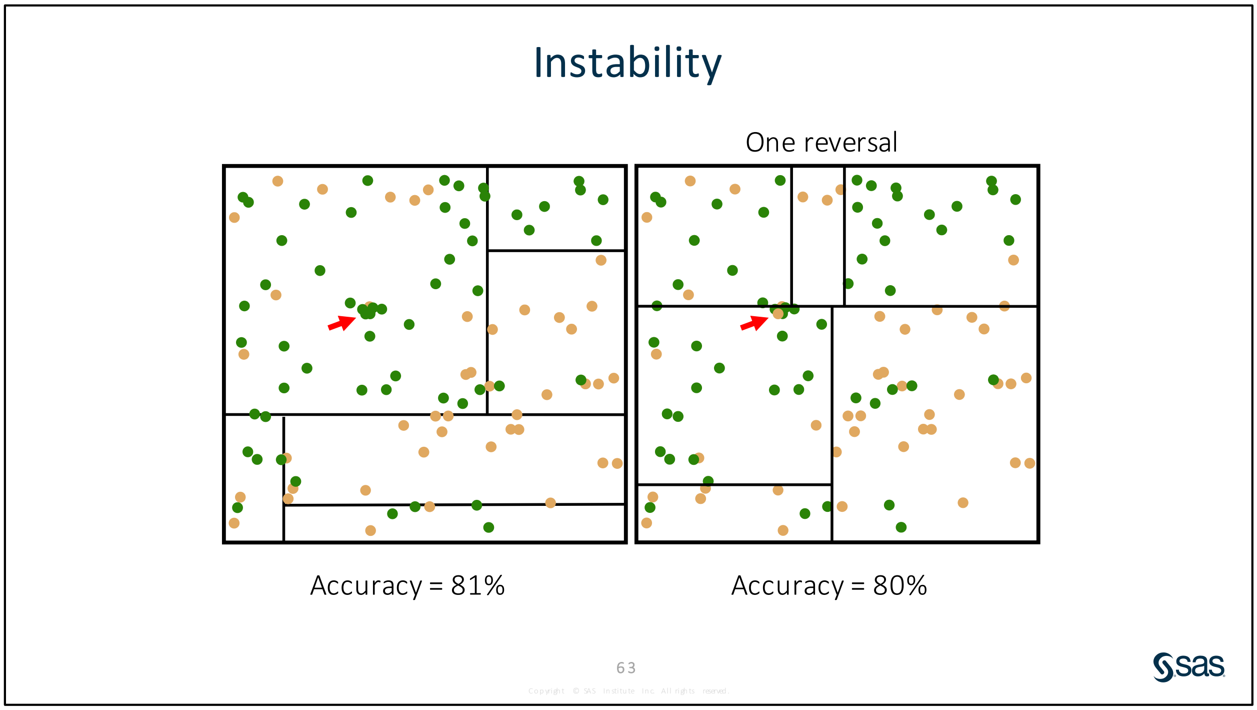

Deployment
Data
Discovery

Essential Discovery Tasks

- Select an algorithm.
- Improve the model.
- Optimize complexity of the model.
- Regularize and tune hyperparameters of the model.
- Build ensemble models.

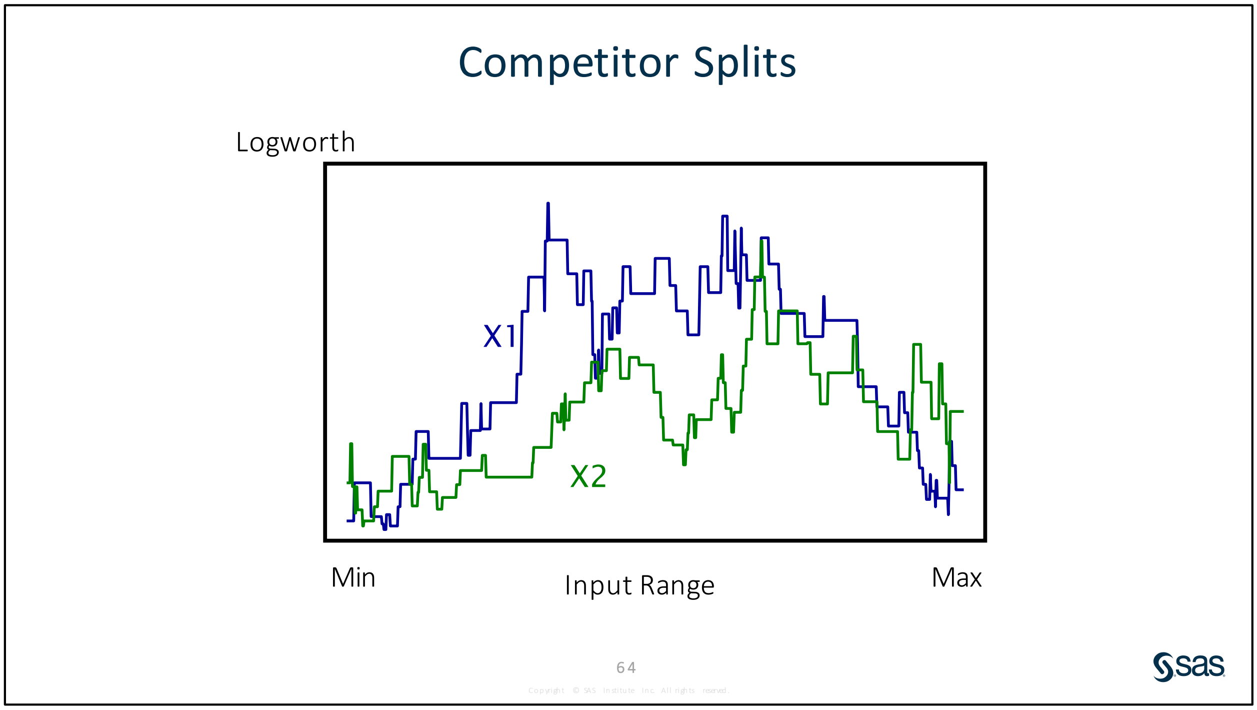
62

Even with an understanding of some of the basic guidelines for selecting an algorithm and incorporating hyperparameter tuning, determining the single most effective machine learning algorithm (and its tuning parameters) to use for a problem and data set is a daunting task. Ensemble modeling can take some of that weight off your shoulders and can give you peace of mind that the predictions are the result of a collaborative effort, or consensus, among multiple models that are trained either from different algorithms that approach the problem from different perspectives, or from the same algorithm applied to different samples or using different tuning parameter settings, or both.



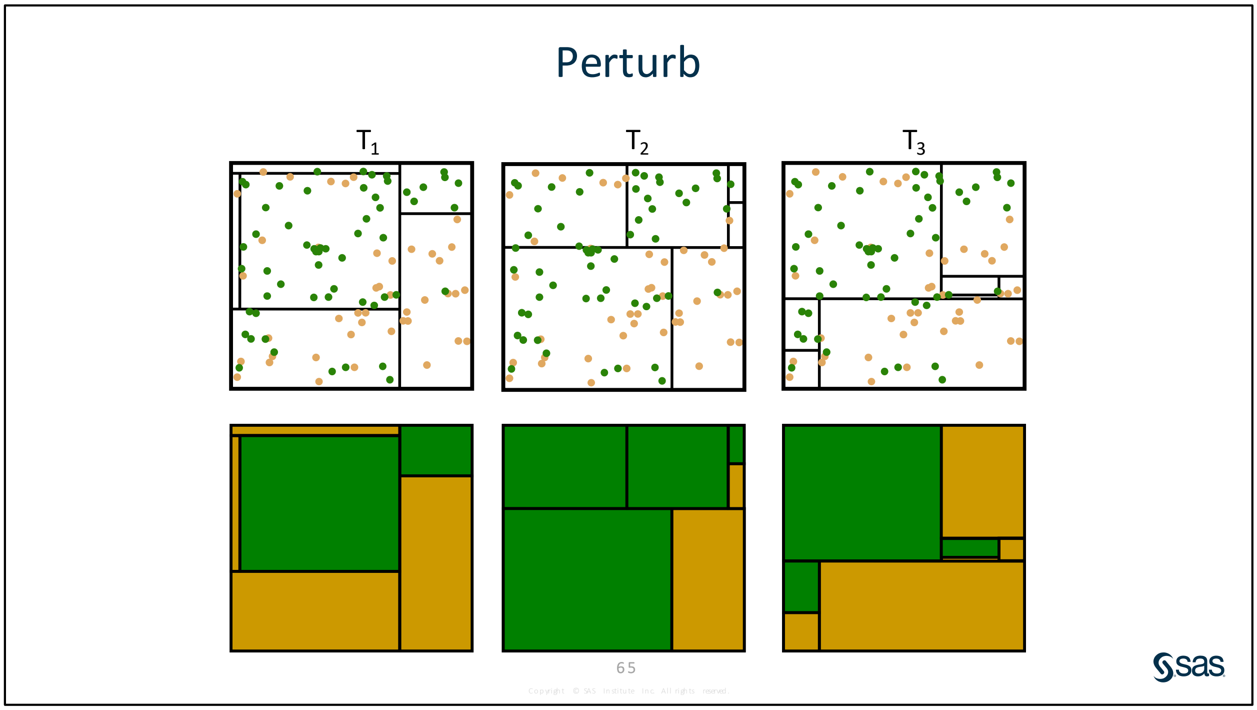
Instability

Decision trees are unstable models. That is, small changes in the training data can cause large changes in the topology of the tree. However, the overall performance of the tree remains stable (Breiman et al. 1984). In the above example, changing the class label of one case resulted in a completely different tree with nearly the same accuracy.



Competitor Splits

The instability results from the large number of univariate splits considered and the fragmentation of the data. At each split, there are typically a number of splits on the same and different inputs that give similar performance (competitor splits). A small change in the data can easily result in a different split being chosen. This in turn produces different subsets in the child nodes. The changes in the data are even larger in the child nodes. The changes continue to cascade down the tree.



Perturb

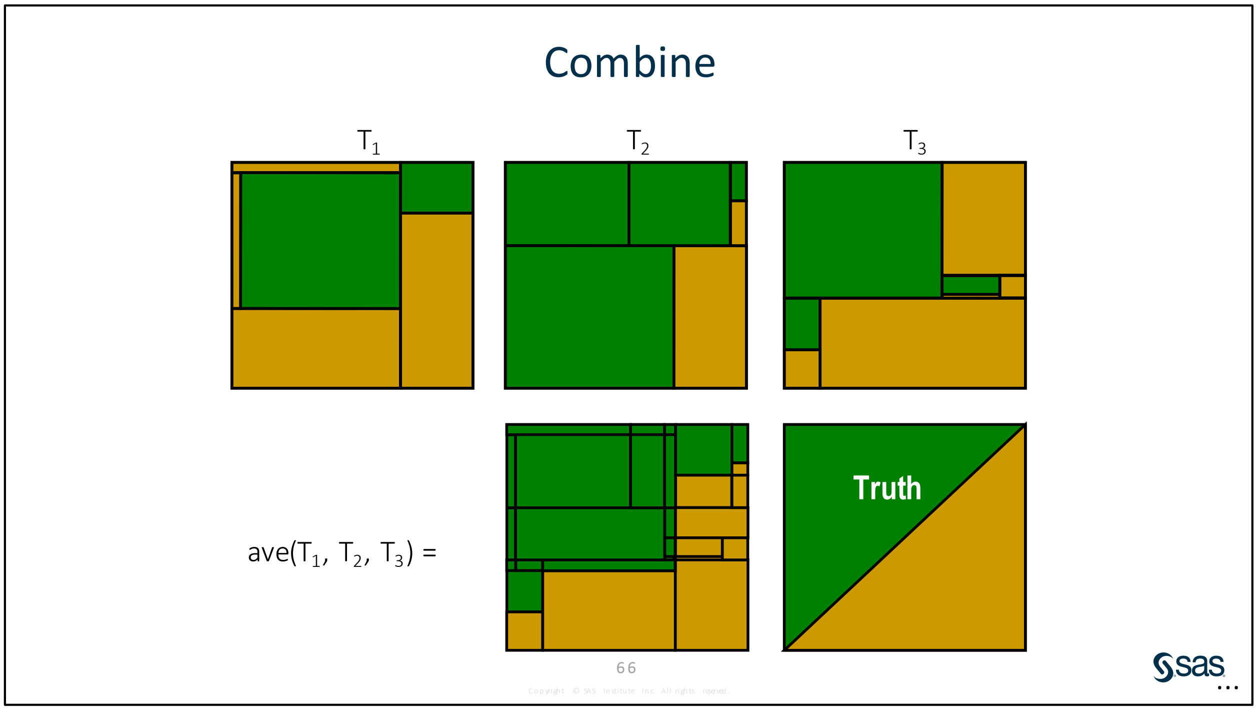
65

Copyright © SAS Institute Inc. All rights reserved.

Methods have been devised to take advantage of the instability of trees to create models that are more powerful. *Perturb and combine* (*P & C*) methods generate multiple models by manipulating the distribution of the data or altering the construction method (such as changing the tree settings) and then averaging the results (Breiman 1998). (The “perturb” step is illustrated above, where perhaps the splitting criteria changes between the trees. The “combine” step is illustrated on the next slide.) Any unstable modeling method can be used, but trees are most often chosen because of their speed and flexibility.

Here are some perturbation methods:

- resample
- subsample
- add noise
- adaptively reweight
- randomly choose from the competitor splits



Combine

T_1
T_2
T_3
ave(T_1, T_2, T_3) =
Truth
66
Copyright © SAS Institute Inc. All rights reserved.
sas

An ensemble model is the combination of multiple models. The combinations can be formed in these ways:

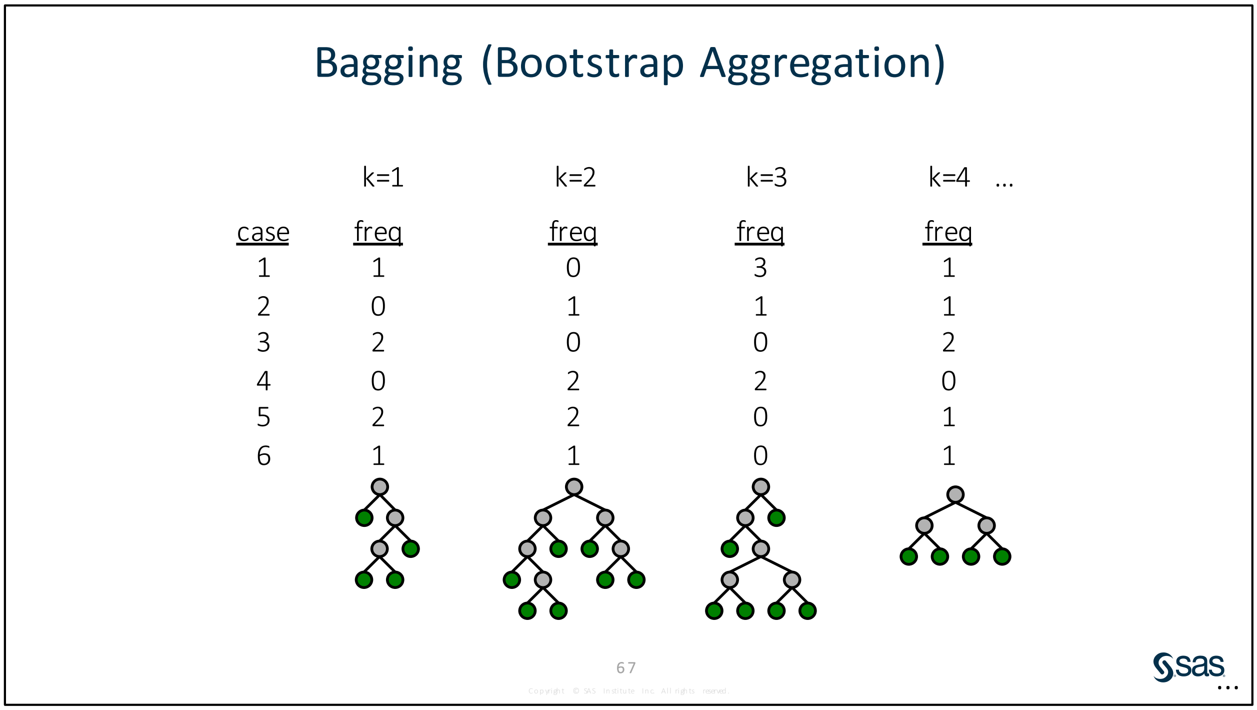
- voting on the classifications
- using weighted voting, where some models have more weight
- averaging (weighted or unweighted) the predicted values

Ensemble methods are a very active area of research in the fields of machine learning and statistics. Many other P & C methods have been devised.

The attractiveness of P & C methods is their improved performance over single models. Bauer and Kohavi (1999) demonstrated the superiority of P & C methods with extensive experimentation. One reason why simple P & C methods give improved performance is variance reduction. If the base models have low bias and high variance, then averaging decreases the variance. In contrast, combining stable models can negatively affect performance. The reasons why adaptive P & C methods work go beyond simple variance reduction and are the topic of much research. (For example, see Breiman 1998.) Graphical explanations show that ensembles of trees have decision boundaries of much finer resolution than would be possible with a single tree (Rao and Potts 1997).

A new case is scored by running it down the multiple trees and averaging the results. Multiple models need to be stored and processed. The simple interpretation of a single tree is lost.

> *Bagging goes a long way towards making a silk purse out of a sow's ear, especially if the sow's ear is twitchy. …What one loses, with the trees, is a simple and interpretable structure. What one gains is increased accuracy.*
>
> — Breiman (1996)



Bagging (Bootstrap Aggregation)

	k=1	k=2	k=3	k=4 …
case	freq	freq	freq	freq
1	1	0	3	1
2	0	1	1	1
3	2	0	0	2
4	0	2	2	0
5	2	2	0	1
6	1	1	0	1

67

…

Bagging (*bootstrap aggregation*) is the original P & C method (Breiman 1996).

1. Draw *K* bootstrap samples.

   A *bootstrap sample* is a random sample of size *n* drawn from the empirical distribution of a sample of size *n*. That is, the training data is resampled with replacement. Some of the cases will be left out of the sample, and some cases will be represented more than once.

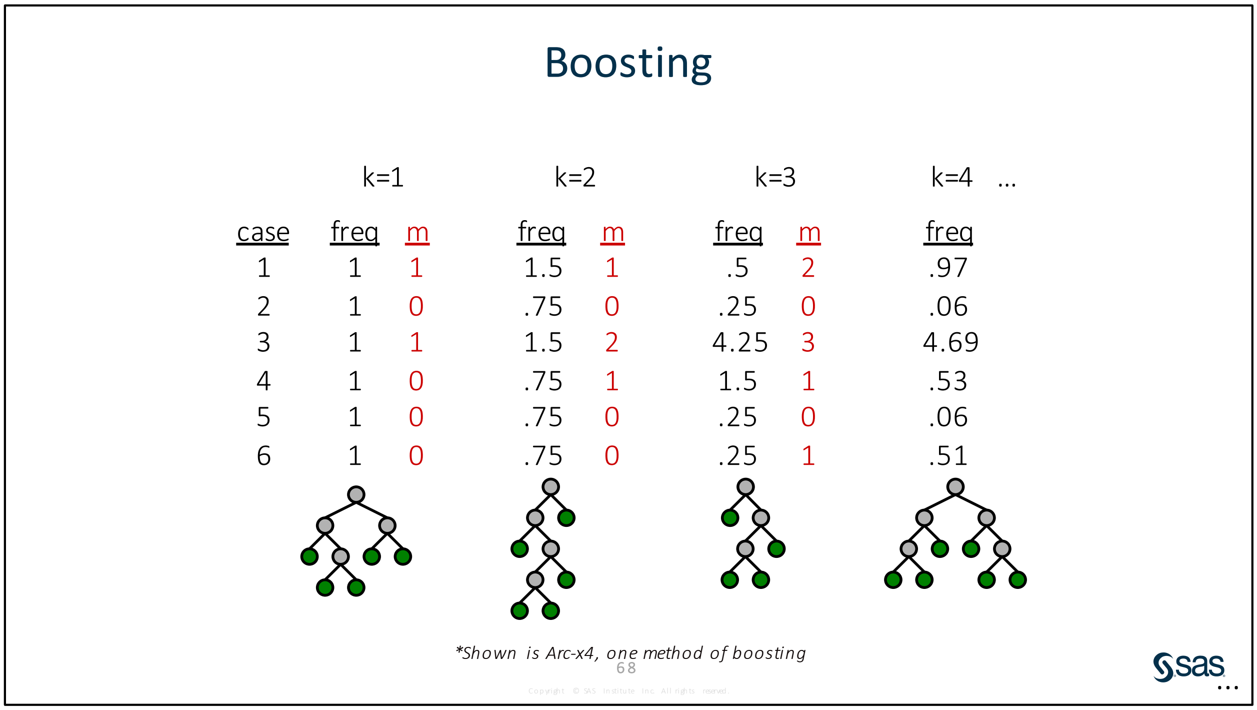
2. Build a tree on each bootstrap sample.

   Pruning can be counterproductive (Bauer and Kohavi 1999). Large trees with low bias and high variance are ideal.

3. Vote or average.

   For classification problems, take the mean of the posterior probabilities or take the plurality vote of the predicted class. Bauer and Kohavi (1999) found that averaging the posterior probabilities gave slightly better performance than voting. Take a mean of the predicted values for regression.

   Breiman (1996) used 50 bootstrap replicates for classification and 25 for regression and for averaging the posterior probabilities. Bauer and Kohavi (1999) used 25 replicates for both voting and averaging.



Boosting

	k=1		k=2		k=3		k=4 …
case	freq	m	freq	m	freq	m	freq
1	1	1	1.5	1	.5	2	.97
2	1	0	.75	0	.25	0	.06
3	1	1	1.5	2	4.25	3	4.69
4	1	0	.75	1	1.5	1	.53
5	1	0	.75	0	.25	0	.06
6	1	0	.75	0	.25	1	.51

*Shown is Arc-x4, one method of boosting

68

"Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance." (Breiman 1996). "The term boosting refers to a family of algorithms that are able to convert weak learners to strong learners." (Zhou 2012).

Arcing (adaptive resampling and combining) methods are examples of boosting. They sequentially perturb the training data based on the results of the previous models. Cases that are incorrectly classified are given more weight in subsequent models. Arc-x4 (Breiman 1998) is a simplified version of the AdaBoost (adaptive boosting, also known as Arc-fs) algorithm of Freund and Schapire (1996). Both algorithms give similar performance (Breiman 1998, Bauer and Kohavi 1999).

At the *k*th step, a model (decision tree) is fit using weights for each case. For the *i*th case, the arc-x4 weights (that is, the selection probabilities) are

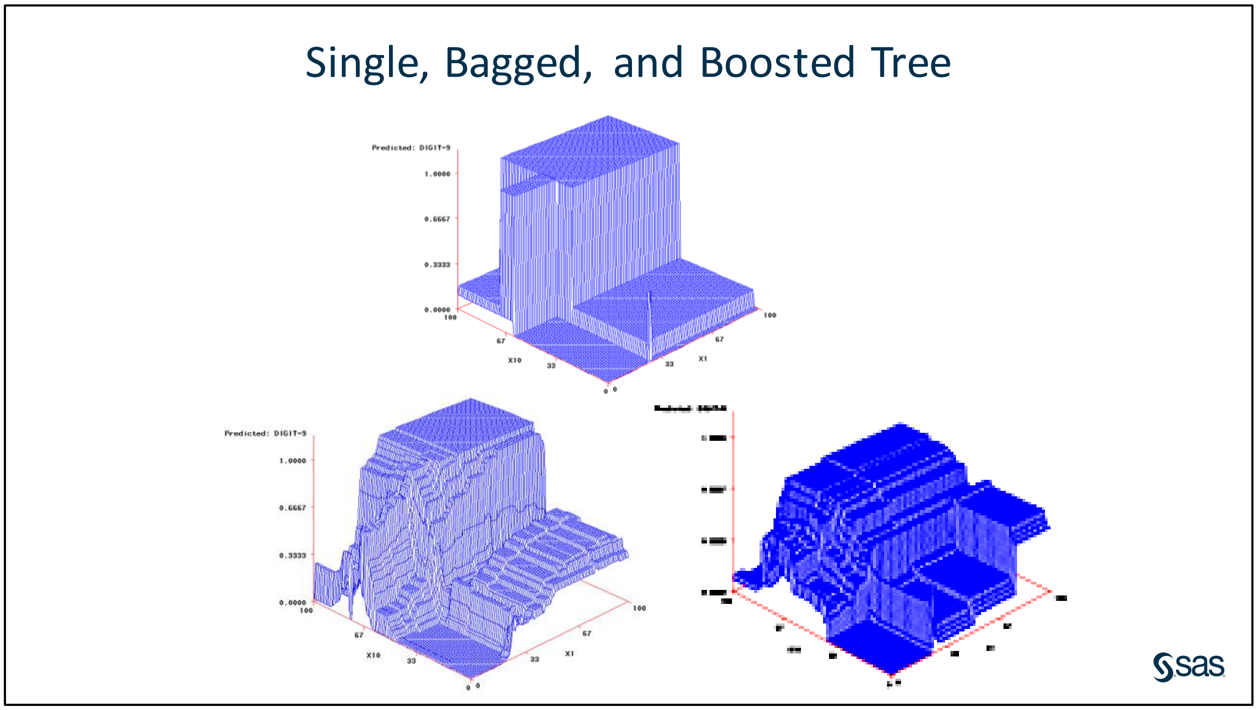
$$p(i) = \frac{1+m(i)^4}{\sum\left(1+m(i)^4\right)},$$

where $0 \le m(i) \le k$ is the number of times that the i^{th} case is misclassified in the preceding steps. Unlike bagging, pruning the individual trees improves performance (Bauer and Kohavi 1999).

The weights are incorporated either by using a weighted analysis or by resampling the data such that the probability that the i^{th} case is selected is $p(i)$. For convenience, the weights can be normalized to frequencies by multiplying by the sample size, *n* (as shown above). Bauer and Kohavi (1999) found that resampling performed better than reweighting for arc-x4 but did not change the performance of AdaBoost. AdaBoost uses a different (more complicated) formula for $p(i)$. Both formulas put greater weight on cases that are frequently misclassified.

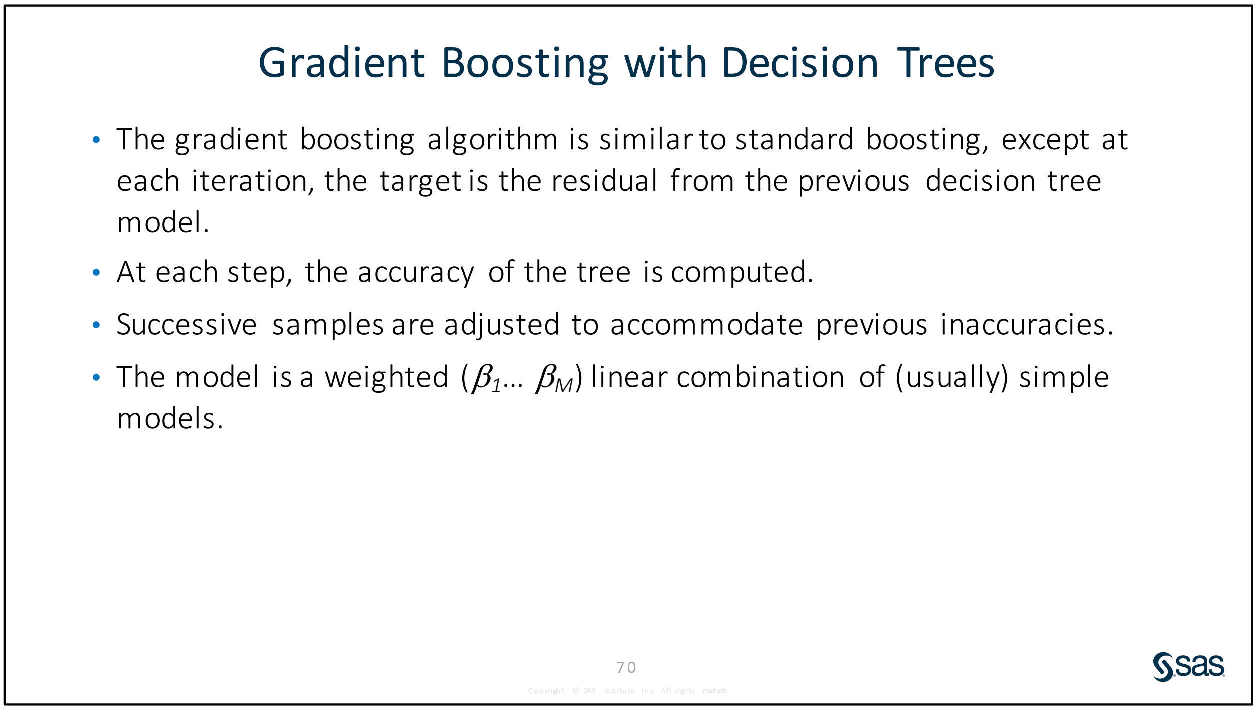
The process is repeated *K* times, and the *K* models are combined by voting or averaging the posterior probabilities. AdaBoost uses weighted voting where models with fewer misclassifications, particularly of the hard-to-classify cases, are given more weight. Breiman (1998) used *K*=50. Bauer and Kohavi (1999) used *K*=25.

Arcing improves performance to a greater degree than bagging, but the improvement is less consistent (Breiman 1998, Bauer and Kohavi 1999).



Single, Bagged, and Boosted Tree

You can visualize the effects of bagging and boosting on a data set with two inputs. These methods tend to smooth the prediction surface when compared to a single tree.



Gradient Boosting with Decision Trees

- The gradient boosting algorithm is similar to standard boosting, except at each iteration, the target is the residual from the previous decision tree model.
- At each step, the accuracy of the tree is computed.
- Successive samples are adjusted to accommodate previous inaccuracies.
- The model is a weighted ($\beta_1 \ldots \beta_M$) linear combination of (usually) simple models.

Details: Gradient Boosting with an Interval Target

The gradient boosting algorithm is a weighted ($\beta_1 \ldots \beta_M$) linear combination of (usually) simple models ($T_1 \ldots T_M$). (Friedman 2001). In SAS Visual Data Mining and Machine Learning, the base model is a decision tree.

Begin with an initial guess, F_0, and proceed in a stage-wise manner fitting subsequent (*m*) tree models to "pseudo" residuals ($\tilde{y}_{im}$). The residuals are computed from target values (y_i) and predictions from the function at the previous iteration ($F_{m-1}(x_i)$). The function $F_m(x)$ is updated by adding the fitted model, $\nu\beta_m T_m(x)$ to $F_{m-1}(x)$.

The shrinkage parameter, ν (0<ν<1) controls the learning rate of the algorithm. Friedman (2001) found that small values (≤ 0.1) lead to better generalization.

In regression trees with interval targets and least-square loss criterion, the "pseudo" residual, $\tilde{y}_{im}$, and the 'guess', F_0 , are defined as follows:

$$\tilde{y}_{im} = y_i - F_{m-1}(x_i)$$

$$F_0 = \bar{y}$$

In classification trees with a binary target ($y \in \{-1,1\}$) and binomial log-likelihood loss criterion, the "pseudo" residuals and F_0 are

$$\tilde{y}_{im} = 2y_i / (1 + \exp(2y_i F_{m-1}(x_i)))$$

$$F_0 = \frac{1}{2}\log\left(\frac{1+\bar{y}}{1-\bar{y}}\right).$$

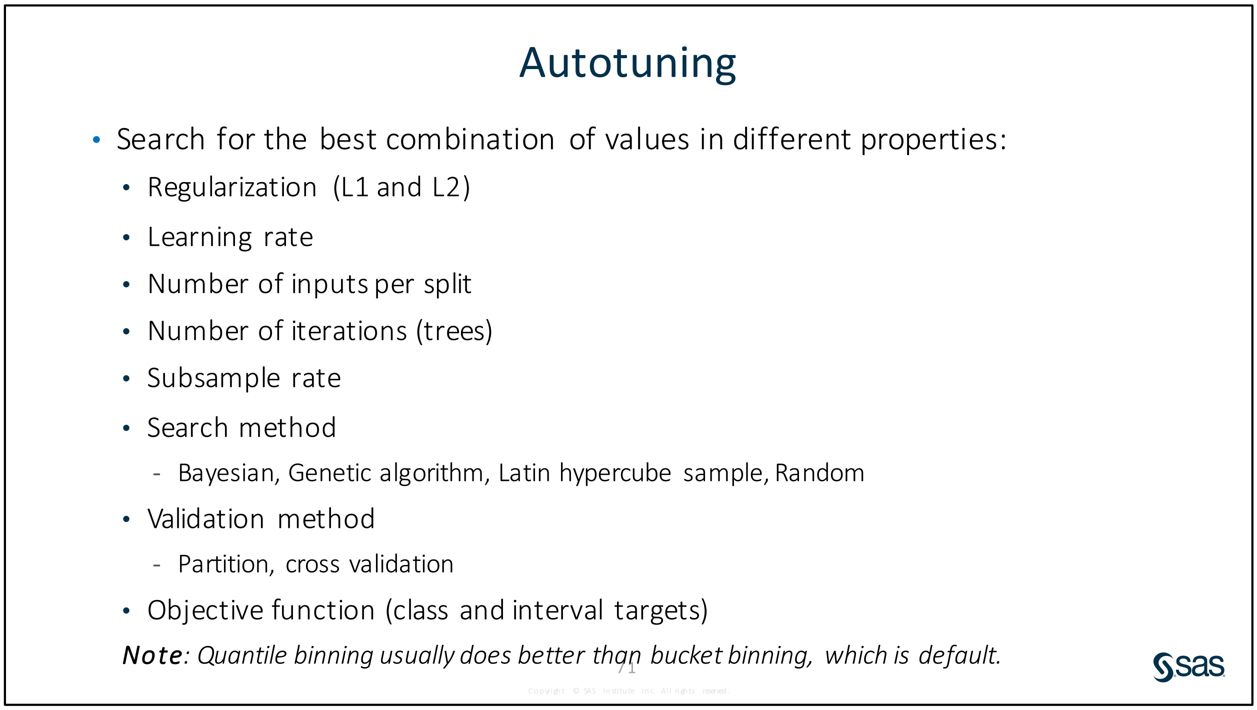
The binary target predictions from the final approximation $F_M(x)$ can be transformed to yield probability estimates.

$$\hat{p}_{+1} = 1/(1+e^{-2F_M(x)}),\ \hat{p}_{-1} = 1/(1+e^{2F_M(x)})$$

Friedman (2002) showed that accuracy and speed can be improved by sub-sampling training data randomly (without replacement) at each iteration leading to the stochastic gradient boosting algorithm.

Gradient boosting trains a sequence of trees over multiple iterations similar to boosting. The main difference is that it minimizes a stochastic gradient descent function when oversampling to reduce the residuals of the model.

SAS Viya Data Mining and Machine Learning creates a series of trees, which form a single model. A tree in the series is fit to the residuals of the prediction from the earlier trees in the series. The residual is defined in terms of the derivative of a loss function. For squared error on an interval target, $\hat{r} = y_i - \hat{y}_i$. Each time that the data are used to grow a tree, the accuracy of the tree is computed. Successive samples are adjusted to accommodate previous inaccuracies. Each successive sample is weighted per the accuracy of the previous models. (See the SAS Viya Data Mining and Machine Learning documentation for more details.)



Autotuning

- Search for the best combination of values in different properties:
  - Regularization (L1 and L2)
  - Learning rate
  - Number of inputs per split
  - Number of iterations (trees)
  - Subsample rate
  - Search method
    - Bayesian, Genetic algorithm, Latin hypercube sample, Random
  - Validation method
    - Partition, cross validation
  - Objective function (class and interval targets)

Note: *Quantile binning usually does better than bucket binning, which is default.*

71

Autotuning searches for the best combination of the gradient boosting parameters. ***Performing autotuning can substantially increase run time***.

Autotuning runs based on the following options, which limit the search of all possible combinations in terms of the gradient boosting parameters.

L1 Regularization penalizes the absolute value for the weights. Different values for L1 are tried between the range defined by From and To. The default initial value for the L1 is 0. The default for the range is from 0 to 10.

L2 Regularization penalizes the square value for the weights. Different values for L2 are tried between the range established by From and To. The default initial value for the L2 is 0. The default for the range is from 0 to 10.

Learning Rate controls the size of the weight changes. It ranges from 0 (exclusive) to 1. The default initial value is 0.1. The default initial value for the learning rate is 0.1. The default for the range is from 0.01 to 1.

Number of Inputs per Split specifies the number of inputs evaluated per split. The default value is 100. The default range is from 1 to100.

Number of Iterations specifies the number of iterations of a boosting series. The default initial value is 100. The range is from 20 to 150.

Subsample Rate specifies the subsample rate. The default initial value is 0.5. The default range is from 0.1 1.

Search Options specifies the options for autotuning searching. The following options are available:

- **Genetic algorithm** uses an initial Latin hypercube sample that seeds a genetic algorithm. The genetic algorithm generates a new population of alternative configurations at each iteration.
- **Latin hypercube sample** performs an optimized grid search that is uniform in each tuning parameter, but random in combinations.
- **Random** generates a single sample of purely random configurations.
- **Bayesian** uses priors to seed the iterative optimization.

Number of evaluations per iteration specifies the number of tuning evaluations in one iteration. This option is available only if the Search method is Genetic algorithm or Bayesian. The default value is 10. It ranges from 2 to 2,147,483,647.

Maximum number of evaluations specifies the maximum number of tuning evaluations. This option is available only if the Search method is Genetic algorithm or Bayesian. The default value is 50. It ranges from 3 to 2,147,483,647.

Maximum number of iterations specifies the maximum number of tuning iterations. This option is available only if the Search method is Genetic algorithm or Bayesian. The default value is 5. It ranges from 1 to 2,147,483,647.

Sample size specifies the sample size. This option is available only if the Search method is Random or Latin hypercube sample. The default value is 50. It ranges from 2 to 2,147,483,647.

There are some general options associated with the autotuning search.

Validation method specifies the validation method for finding the objective value. If your data is partitioned, then that partition is used. Validation method, Validation data proportion, and Cross validation number of folds are all ignored.

- **Partition** specifies using the partition validation method. With partition, you specify proportions to use for randomly assigning observations to each role.
  - **Validation data proportion** specifies the proportion of data to be used for the Partition validation method. The default value is 0.3.
- **K-fold cross validation** specifies using the cross validation method. In cross validation, each model evaluation requires k training executions (on k-1 data folds) and k scoring executions (on one holdout fold). This increases the evaluation time by approximately a factor of k.
  - **Cross validation number of folds** specifies the number of partition folds in the cross validation process (the k defined above). Possible values range from 2 to 20. The default value is 5.

Nominal target objective function specifies the objective function to optimize for tuning parameters for a nominal target. Possible values are average squared error, area under the curve, F1 score, F0.5 score, gamma, Gini coefficient, Kolmogorov-Smirnov statistic, multi-class log loss, misclassification rate, root average squared error, and Tau. The default value is misclassification rate.

Interval target objective function specifies the objective function to optimize for tuning parameters for an interval target. Possible values are average square error, mean absolute error, mean square logarithmic error, root average square error, root mean absolute error, and root mean square logarithmic error. The default value is average square error.

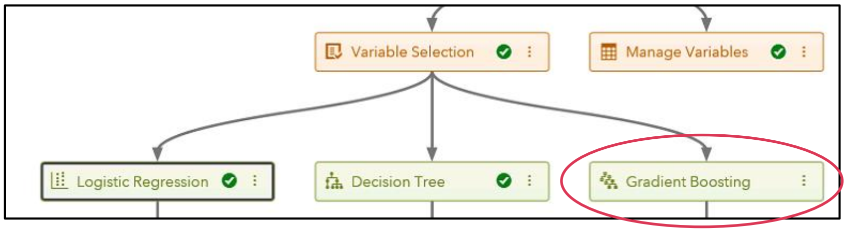
Building a Gradient Boosting Model

The algorithm for gradient boosting evolved from the application of boosting methods to regression trees. The main idea is to compute a sequence of simple trees, where each successive tree is built for the prediction residuals of the preceding tree. This method builds trees by partitioning the data into samples at each split node. Then, at each step of the boosting trees algorithm, a best partitioning of the data is determined, and the deviations of the observed values from the respective residuals for each partition are computed. The next trees will be fitted to those residuals, to find another partition that further reduces the residual variance for the data, given the preceding sequence of trees.

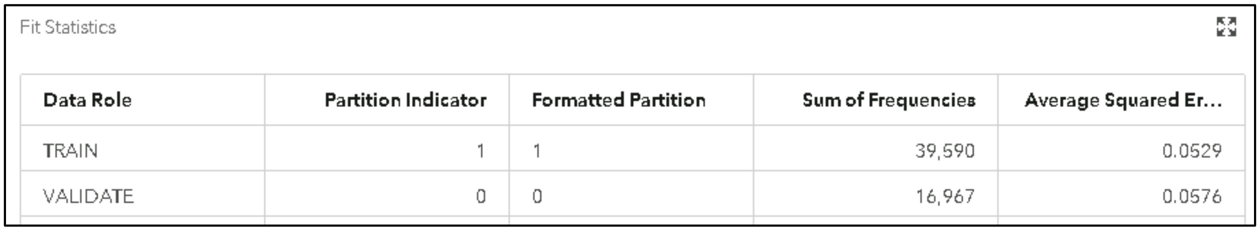
This additive weighted approach of trees can produce excellent fit of the predicted values to the observed values, even if the specific nature of the relationships between the inputs and the target is complex. For that reason, the method of gradient boosting by fitting a weighted additive expansion of simple trees can create general and powerful machine learning models.

In this demonstration, you add a Gradient Boosting node to the Starter Template pipeline. You build a default Gradient Boosting model, change some of the settings, and compare the model to the other models in the pipeline.

1. In the Chapter 3 pipeline, right-click the **Variable Selection** node and select **Add below ⇨ Supervised Learning ⇨ Gradient Boosting**.



2. Keep all properties for the Gradient Boosting node at their defaults. Run the Gradient Boosting node.
3. Open the results for the Gradient Boosting node.
4. Click the **Assessment** tab.



Fit Statistics

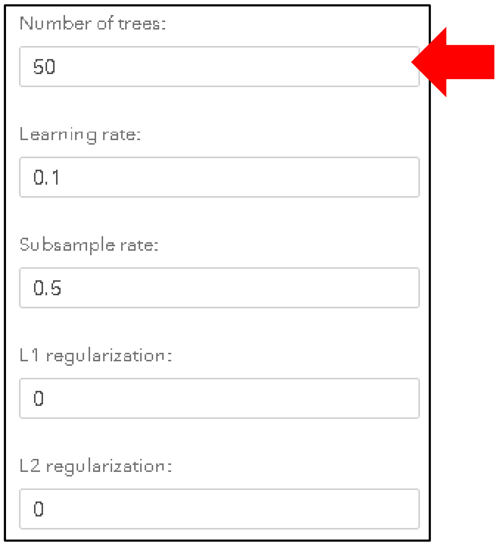
Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.0529
VALIDATE	0	0	16,967	0.0576

The Fit Statistics table, shown above, shows an average square error of 0.0576 on the VALIDATE partition.

This performance is pretty good, even better than the Decision Tree tuned in the previous demonstrations. Regardless, try to improve the Gradient Boosting performance by changing some of the default settings.

5. Close the Results window.
6. Reduce **Number of trees** from **100** to **50**.
7. Under the Tree-splitting Options properties, increase **Maximum depth** from **6** to **8**.

8. Increase **Minimum leaf size** from **5** to **15**.
9. Increase **Number of interval bins** from **20** to **100**.



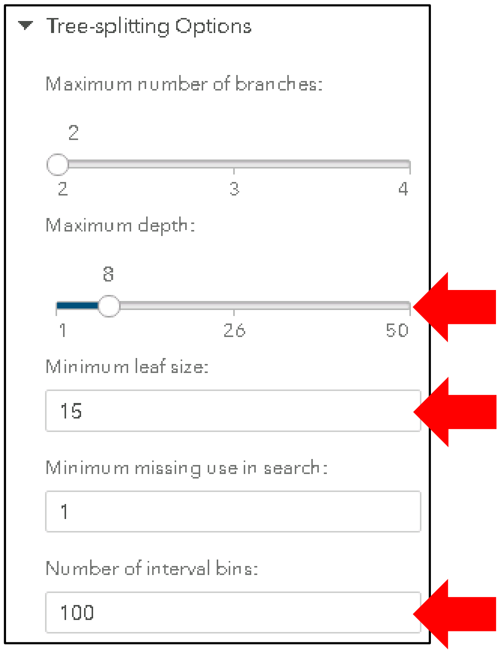
Number of trees: 50

Learning rate: 0.1

Subsample rate: 0.5

L1 regularization: 0

L2 regularization: 0



Tree-splitting Options

Maximum number of branches: 2

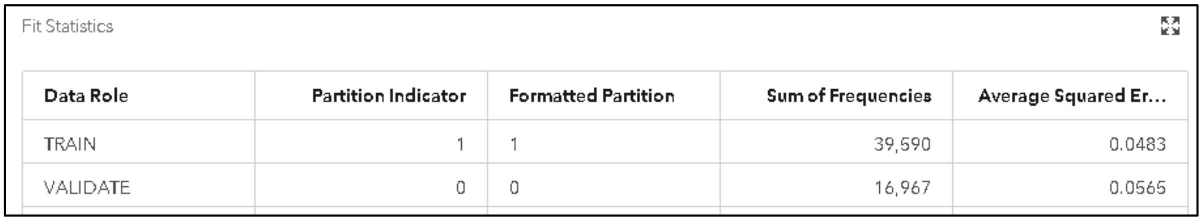
Maximum depth: 8

Minimum leaf size: 15

Minimum missing use in search: 1

Number of interval bins: 100

10. Run the Gradient Boosting node.
11. Open the results for the Gradient Boosting node.



Fit Statistics

Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.0483
VALIDATE	0	0	16,967	0.0565

12. The average square error for the tuned Gradient Boosting model is 0.0565 on the VALIDATE partition. This fit statistic is slightly better than the first model by using the default settings, by approximately 2%.
13. Close the Results window.

End of Demonstration

Exercises

2. Building a Gradient Boosting Model

a. Build a gradient boosting model using the Autotune feature. Add a Gradient Boosting node to the Chapter 3 pipeline, below the Variable Selection node. Use the Autotune feature. Explore the settings that are made available when Autotune is selected. Run a few options by changing the range of the parameters search.

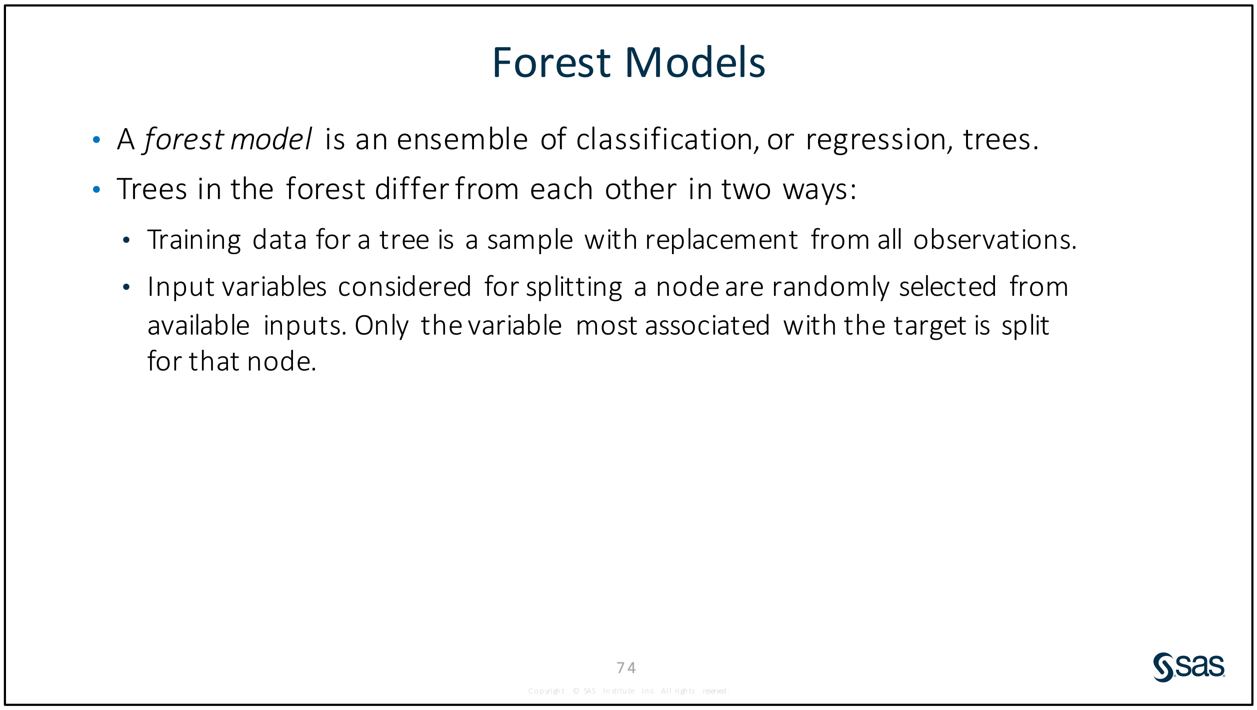
Note: This exercise might take several minutes to run.

b. What criteria were selected for the champion model?

- The number of trees
- Number of variables per split
- Number of bins
- Maximum number of branches
- Maximum depth

c. How does the autotuned Gradient Boosting compare to the other models in the pipeline, particularly to the Gradient Boosting model built during the demonstration? Consider the fit statistic average square error for this comparison.

End of Exercises



Forest Models

- A *forest model* is an ensemble of classification, or regression, trees.
- Trees in the forest differ from each other in two ways:
  - Training data for a tree is a sample with replacement from all observations.
  - Input variables considered for splitting a node are randomly selected from available inputs. Only the variable most associated with the target is split for that node.

74

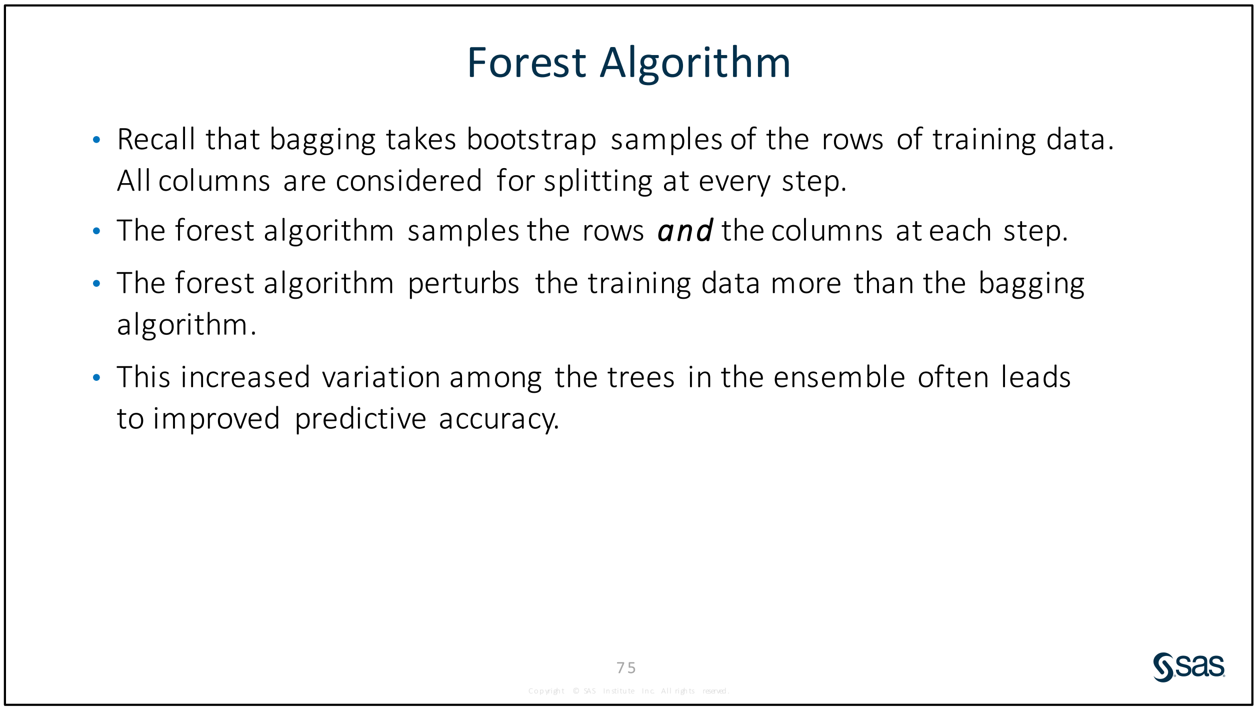
A *random forest* is an ensemble of simple decision trees, each one able to produce its own response to a set of input variables. For classification problems, this response takes the form of a class, which classifies a set of independent variables with one of the categories in the dependent variable. Alternatively, for regression problems, the tree takes the form of an estimate of the dependent variable given the set of independent variables.

A forest model consists of an arbitrary number of simple decision trees that are used to determine the final outcome. For a categorical target, the response of the ensemble of simple decision trees is the vote for the most popular class or the average of the posterior probabilities of the individual trees. For an interval target, the response of the ensemble model is the average of the estimate of the individual decision trees.

The trees that make up a forest differ from each other in two ways:

- The training data for each tree are sampled with replacement from all observations that were originally in the training data.
- The input variables considered for splitting for any given tree are selected randomly from all available inputs.

Among these variables, only the variable most associated with the target is used when forming a split. This means that each tree is created on a sample of the inputs and from a sample of observations. Repeating this process many, many times creates a more stable model than a single tree. The reason for using a sample of the data to construct each tree is because when less than all available observations are used, the generalization error is often improved. In addition, a different sample is taken for each tree.

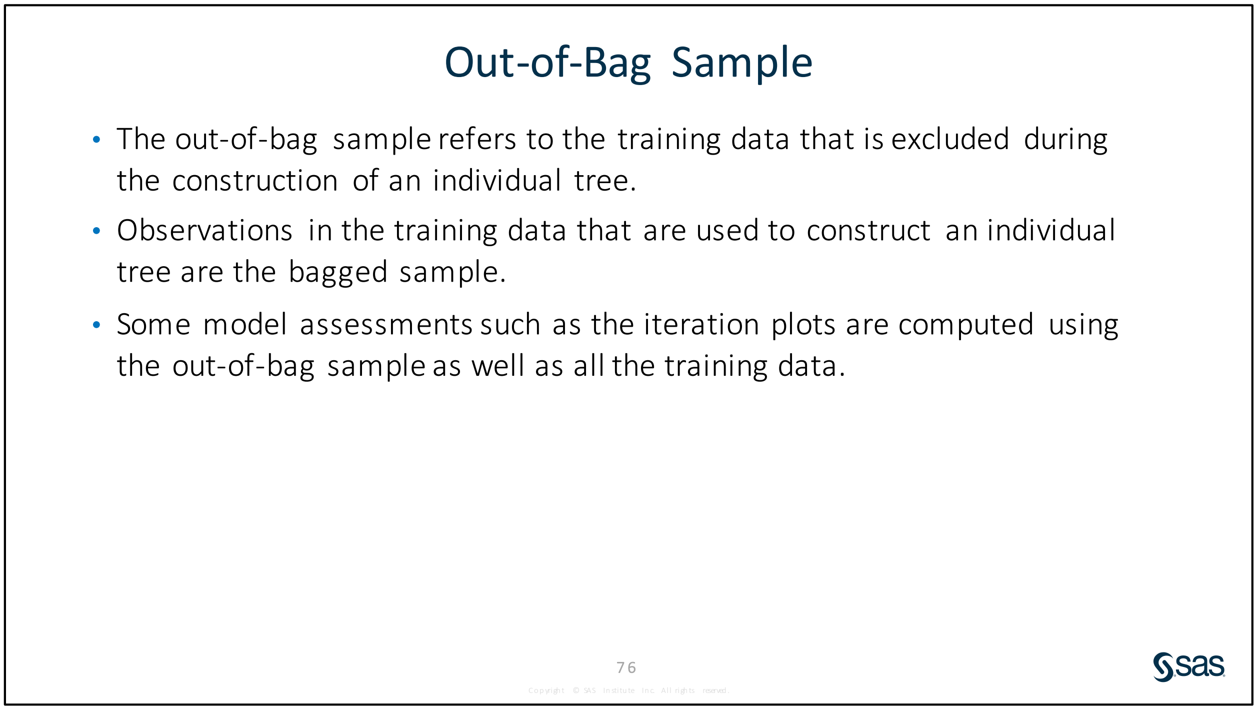


Forest Algorithm

- Recall that bagging takes bootstrap samples of the rows of training data. All columns are considered for splitting at every step.
- The forest algorithm samples the rows ***and*** the columns at each step.
- The forest algorithm perturbs the training data more than the bagging algorithm.
- This increased variation among the trees in the ensemble often leads to improved predictive accuracy.

75

In the forest algorithm, rather than taking bootstrap samples of only the rows, variables are also randomly sampled. This results in a forest, consisting of trees that use different combinations of rows and variables to determine splits. This additional perturbation (beyond bagging) leads to greater diversity in the trees, and better predictive accuracy.



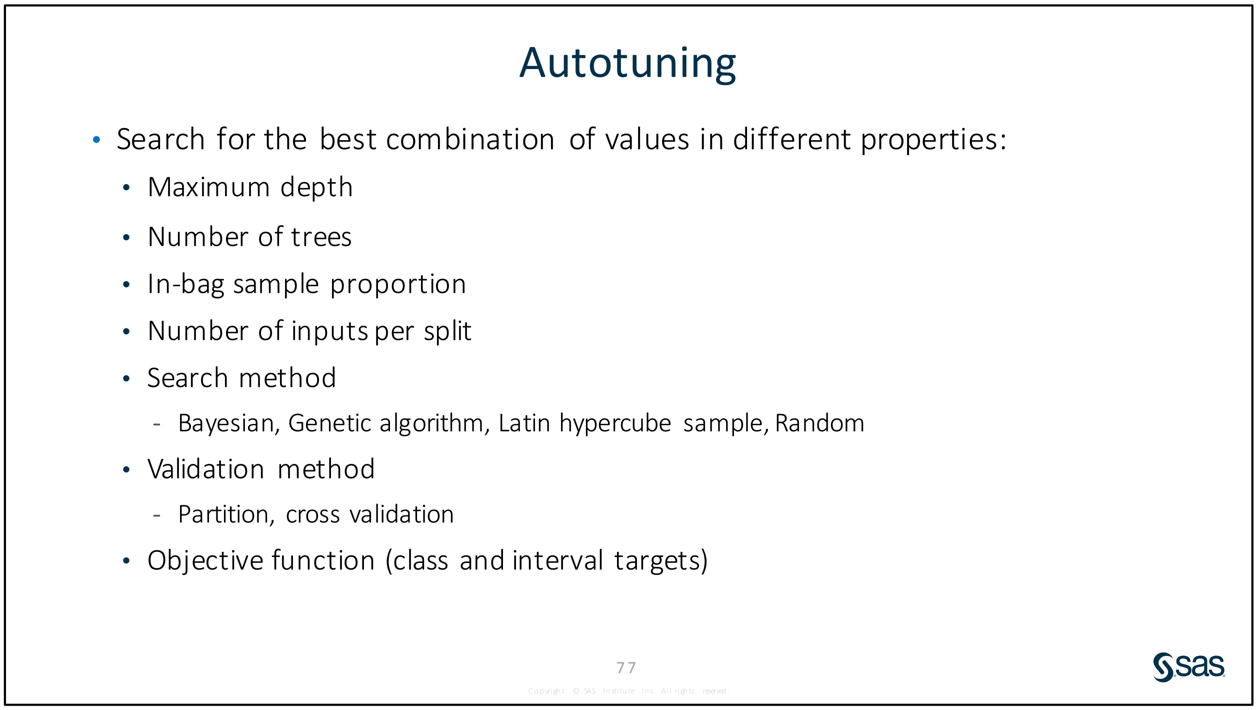
Out-of-Bag Sample

- The out-of-bag sample refers to the training data that is excluded during the construction of an individual tree.
- Observations in the training data that are used to construct an individual tree are the bagged sample.
- Some model assessments such as the iteration plots are computed using the out-of-bag sample as well as all the training data.

76

A decision tree in a forest trains on new training data that are derived from the original training data presented to the model. Training different trees with different training data reduces the correlation of the predictions of the trees, which in turn should improve the predictions of the forest. The training data for an individual tree exclude some of the available data. The data that are withheld from training are called the *out-of-bag sample*. Observations in the training sample are called the *bagged* observations, and the training data for a specific decision tree are called the *bagged data*. For each individual tree, the out-of-bag sample is used to form predictions. These predictions are more reliable than those from training data.

Model assessment such as misclassification rates, average squared error, and iteration plots are constructed on both the entire training data set as well as the out-of-bag sample.



Autotuning

- Search for the best combination of values in different properties:
  - Maximum depth
  - Number of trees
  - In-bag sample proportion
  - Number of inputs per split
  - Search method
    - Bayesian, Genetic algorithm, Latin hypercube sample, Random
  - Validation method
    - Partition, cross validation
  - Objective function (class and interval targets)

77

Autotuning searches for the best combination of the forest parameters. ***Performing autotuning can substantially increase run time***.

Autotuning runs based on some options, which limit the search of all possible combinations in terms of the forest parameters.

Maximum Depth specifies how deep each tree can grow. It ranges from 1 to 50. The default initial value for the maximum depth is 20. The default for the range is from 1 to 29.

Number of Trees specifies the number of trees in the forest. It ranges from 1 to 1000. The default initial value for the number of trees is 100. The default for the range is from 20 to 150.

In-bag Sample Proportion specifies the in-bag sample proportion. It ranges from 0 (exclusive) to 1. The default initial value for the proportion is 0.6. The default for the range is from 0.1 to 0.9.

Number of Inputs per Split specifies the number of inputs evaluated per split. The default value is 100. The default range is from 1 to100.

Search Options specifies the options for autotuning searching. The following options are available:

- **Genetic algorithm** uses an initial Latin hypercube sample that seeds a genetic algorithm. The genetic algorithm generates a new population of alternative configurations at each iteration.
- **Latin hypercube sample** performs an optimized grid search that is uniform in each tuning parameter, but random in combinations.
- **Random** generates a single sample of purely random configurations.
- **Bayesian** uses priors to seed the iterative optimization.

Number of evaluations per iteration specifies the number of tuning evaluations in one iteration. This option is available only if the search method is Genetic algorithm or Bayesian. The default value is 10. It ranges from 2 to 2,147,483,647.

Maximum number of evaluations specifies the maximum number of tuning evaluations. This option is available only if the search method is Genetic algorithm or Bayesian. The default value is 50. It ranges from 3 to 2,147,483,647.

Maximum number of iterations specifies the maximum number of tuning iterations. This option is available only if the search method is Genetic algorithm or Bayesian. The default value is 5. It ranges from 1 to 2,147,483,647.

Sample size specifies the sample size. This option is available only if the search method is Random or Latin hypercube sample. The default value is 50. It ranges from 2 to 2,147,483,647.

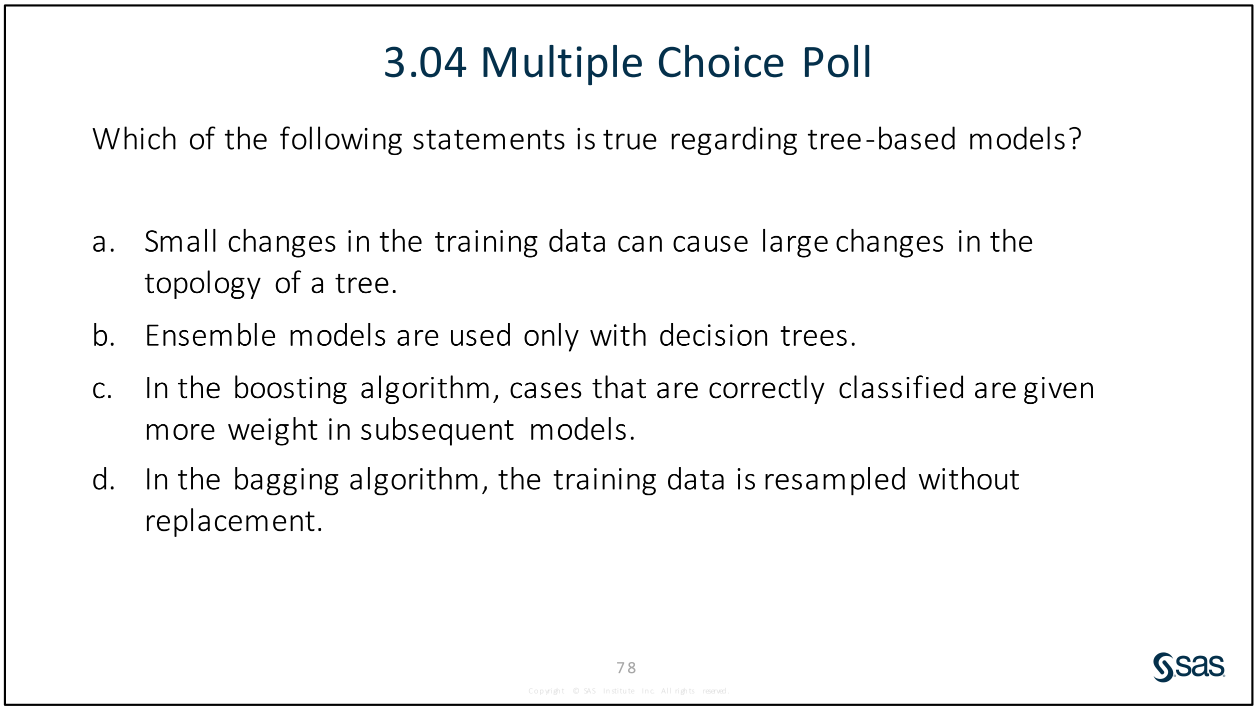
There are some general options associated with the autotuning search.

Validation method specifies the validation method for finding the objective value. If your data is partitioned, then that partition is used. Validation method, Validation data proportion, and Cross validation number of folds are all ignored.

- **Partition** specifies using the partition validation method. With partition, you specify proportions to use for randomly assigning observations to each role.
  - **Validation data proportion** specifies the proportion of data to be used for the Partition validation method. The default value is 0.3.
- **K-fold cross validation** specifies using the cross validation method. In cross validation, each model evaluation requires k training executions (on k-1 data folds) and k scoring executions (on one holdout fold). This increases the evaluation time by approximately a factor of k.
  - **Cross validation number of folds** specifies the number of partition folds in the cross validation process (the k defined above). Possible values range from 2 to 20. The default value is 5.

Nominal target objective function specifies the objective function to optimize for tuning parameters for a nominal target. Possible values are average square error, area under the curve, F1 score, F0.5 score, gamma, Gini coefficient, Kolmogorov-Smirnov statistic, multi-class log loss, misclassification rate, root average square error, and Tau. The default value is misclassification rate.

Interval target objective function specifies the objective function to optimize for tuning parameters for an interval target. Possible values are average square error, mean absolute error, mean square logarithmic error, root average square error, root mean absolute error, and root mean square logarithmic error. The default value is average square error.



3.04 Multiple Choice Poll

Which of the following statements is true regarding tree-based models?

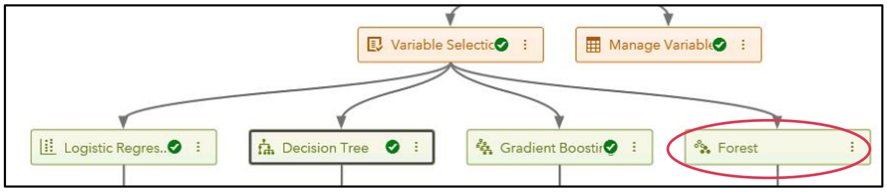
a. Small changes in the training data can cause large changes in the topology of a tree.
b. Ensemble models are used only with decision trees.
c. In the boosting algorithm, cases that are correctly classified are given more weight in subsequent models.
d. In the bagging algorithm, the training data is resampled without replacement.

78

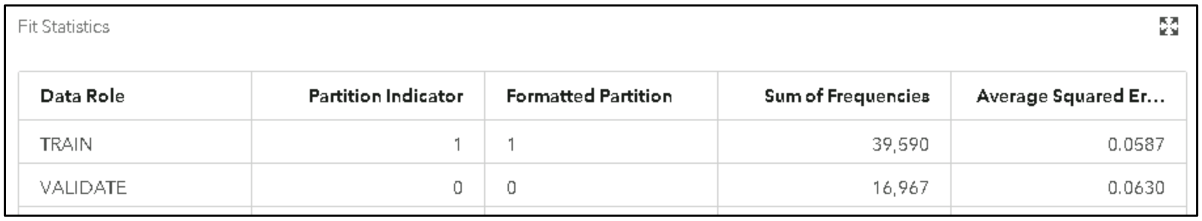
Modeling a Binary Target with a Forest

In this demonstration, you add a Forest node to the Chapter 3 pipeline. You build a default Forest model, change some of the settings, and compare the model to the other models in the pipeline.

1. In the Starter Template pipeline, right-click the **Variable Selection** node and select **Add below ⇨ Supervised Learning ⇨ Forest**.



2. Keep all properties for the Forest at their defaults. Run the Forest node.
3. Open the results for the Forest node.
4. Click the **Assessment** tab.



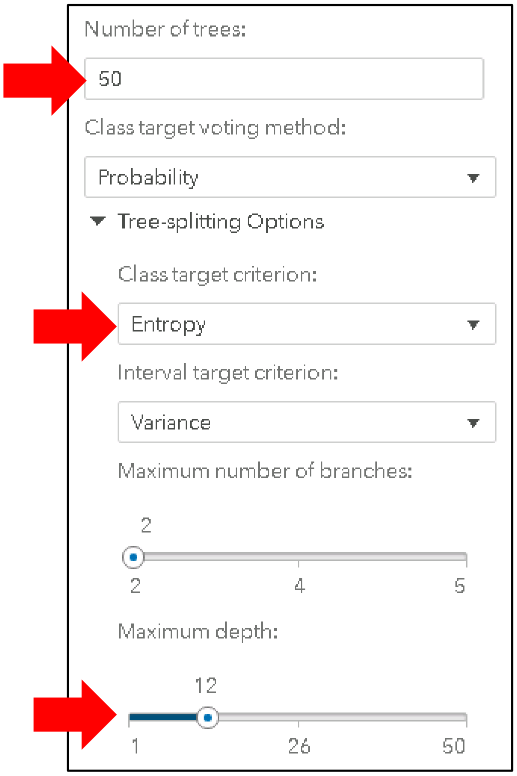
Fit Statistics

Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.0587
VALIDATE	0	0	16,967	0.0630

The Fit Statistics table shows an average square error of 0.0630 on the VALIDATE partition.

5. This performance is again pretty good, better than the decision tree tuned in the previous demonstrations. But again, try to improve the forest performance by changing some of the default settings. Close the Results window.
6. Reduce **Number of trees** from **100** to **50**.
7. Under the Tree-splitting Options properties, change **Class Target Criterion** from **Information gain ratio** to **Entropy**.
8. Decrease **Maximum depth** from **20** to **12**.
9. Increase **Minimum leaf size** from **5** to **15**.
10. Increase **Number of interval bins** from **20** to **100**.

11. Clear the box for the option **Use default number of inputs to consider per split**, which by default is the square root of the number of available inputs. Set this parameter to **7** (half of inputs).



Number of trees: 50

Class target voting method: Probability

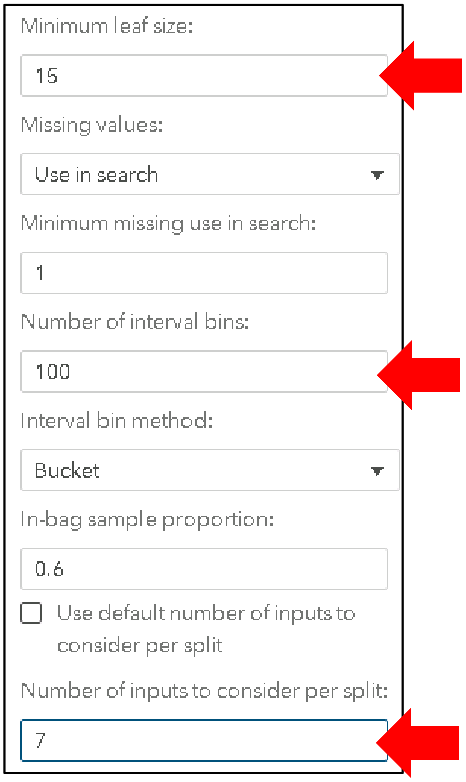
▼ Tree-splitting Options

Class target criterion: Entropy

Interval target criterion: Variance

Maximum number of branches: 2

Maximum depth: 12



Minimum leaf size: 15

Missing values: Use in search

Minimum missing use in search: 1

Number of interval bins: 100

Interval bin method: Bucket

In-bag sample proportion: 0.6

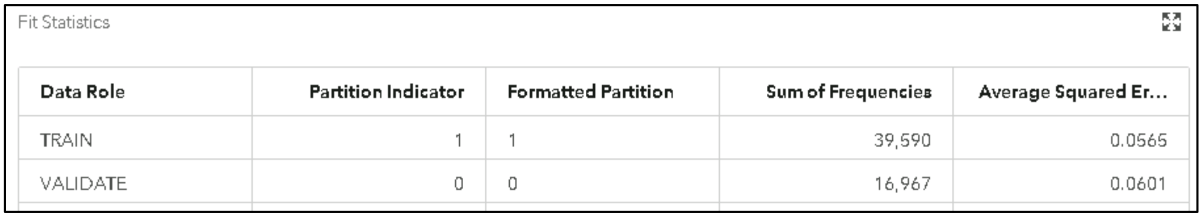
☐ Use default number of inputs to consider per split

Number of inputs to consider per split: 7

12. Run the Forest node.

13. Open the results for the Forest node.

14. Click the **Assessment** tab.



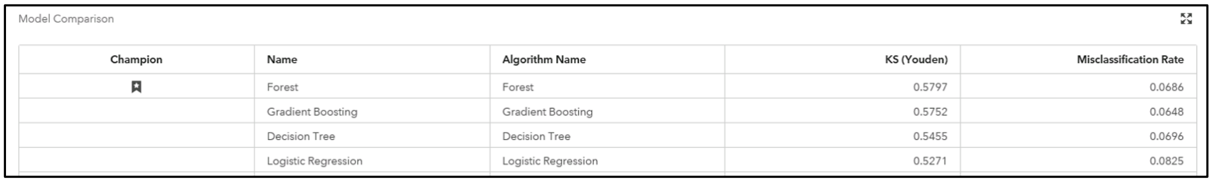
Fit Statistics

Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.0565
VALIDATE	0	0	16,967	0.0601

The average square error for the tuned Forest model is 0.0601 on the VALIDATE partition. This fit statistic is a little bit better than the first model by using the default settings, by approximately 5%.

15. Close the Results window.

16. Run the entire pipeline and view the results of model comparison. The Forest is now the champion model of the pipeline, based on default KS.



Model Comparison

Champion	Name	Algorithm Name	KS (Youden)	Misclassification Rate
🔖	Forest	Forest	0.5797	0.0686
	Gradient Boosting	Gradient Boosting	0.5752	0.0648
	Decision Tree	Decision Tree	0.5455	0.0696
	Logistic Regression	Logistic Regression	0.5271	0.0825

17. Close the Results window.

End of Demonstration

Exercises

3. Building a Forest Model

a. Build a forest using the Autotune feature. Add a Forest node to the Chapter 3 pipeline, below the Variable Selection node. Use the Autotune feature. Explore the settings that are available when Autotune is selected.

Note: This exercise might take several minutes to run.

b. What criteria were selected for the champion model?

- The number of trees
- Number of variables per split
- Number of bins
- Maximum number of branches
- Maximum depth

c. How does the autotuned forest compare to the other models in the pipeline, particularly to the Forest model built during the demonstration? Consider the fit statistic average square error for this comparison.

End of Exercises

3.6 Solutions

Solutions to Exercises

1. **Building a Decision Tree**

   a. Build a decision tree using the Autotune feature. Add a Decision Tree node to the Chapter 3 pipeline, below to the Variable Selection node. Use the Autotune feature. Explore the settings that are made available when **Autotune** is selected, but keep all properties at their defaults.

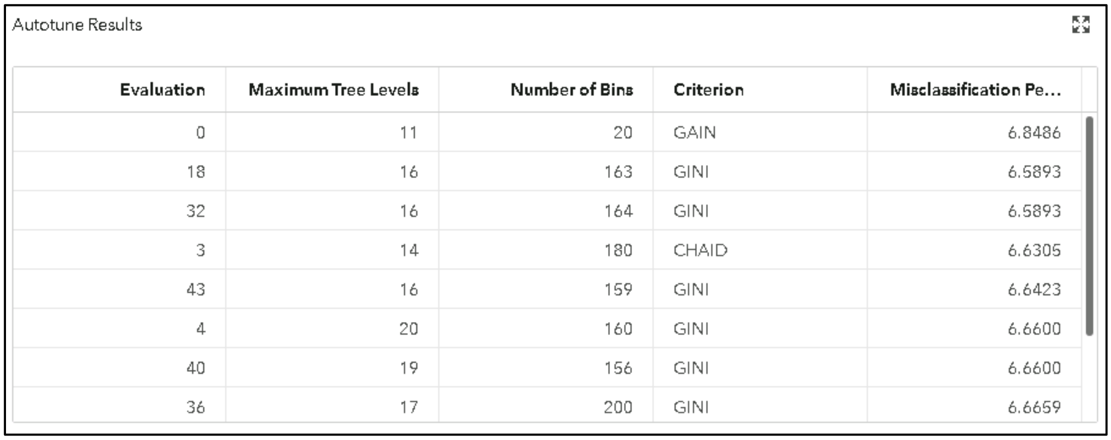
   1) On Starter Template pipeline, right-click the **Variable Selection** node and select **Add below ⇨ Supervised Learning ⇨ Decision Tree**.

   2) In the properties pane, turn on the **Perform Autotuning** option. The default properties show starting values and ranges that are tried for each property in the Decision Tree model.

   3) Right-click the **Decision Tree** node and select **Run**. This process might take few minutes.

   4) When the execution is over, right-click the **Decision Tree** node and select **Results**.

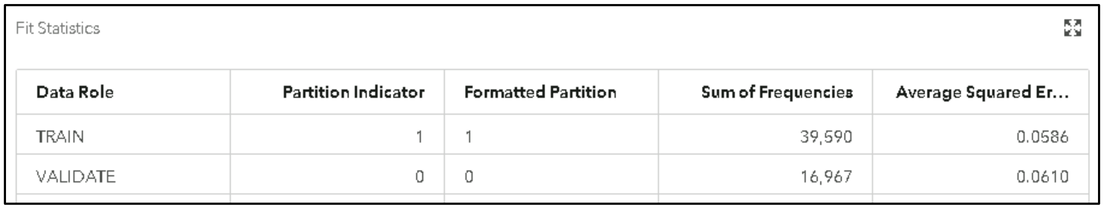
   5) Examine the Results window. Maximize the Autotune Results window and notice the different evaluations performed. Restore the Autotune Results window.



Autotune Results

Evaluation	Maximum Tree Levels	Number of Bins	Criterion	Misclassification Pe...
0	11	20	GAIN	6.8486
18	16	163	GINI	6.5893
32	16	164	GINI	6.5893
3	14	180	CHAID	6.6305
43	16	159	GINI	6.6423
4	20	160	GINI	6.6600
40	19	156	GINI	6.6600
36	17	200	GINI	6.6659

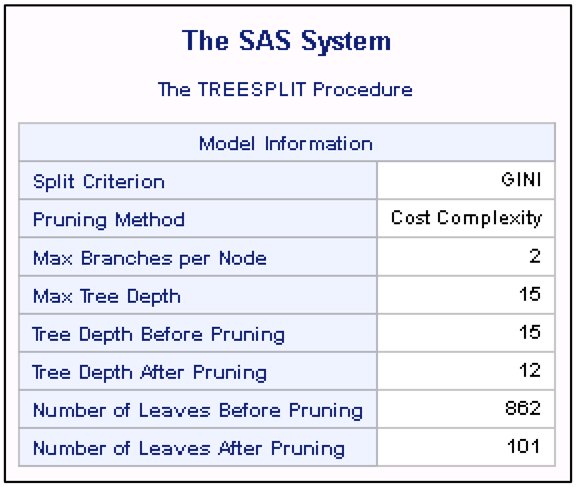
   6) Scroll down and observe the Fit Statistics window. The average square error for the Autotune model is 0.0610 on the VALIDATE partition.



Fit Statistics

Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.0586
VALIDATE	0	0	16,967	0.0610

7) Scroll down and maximize the Output window. This output shows the set of parameters selected for the final Decision Tree model.



The SAS System

The TREESPLIT Procedure

Model Information	
Split Criterion	GINI
Pruning Method	Cost Complexity
Max Branches per Node	2
Max Tree Depth	15
Tree Depth Before Pruning	15
Tree Depth After Pruning	12
Number of Leaves Before Pruning	862
Number of Leaves After Pruning	101

b. What criteria were selected for the champion model?

- Split criteria: **GINI**
- Pruning method: **Cost Complexity**
- Maximum number of branches: **2**
- Maximum tree depth: **15**

c. How does the autotuned decision tree compare to the other models in the pipeline, particularly to the Decision Tree model built during the demonstration? Consider the fit statistics average square error for this comparison.

It performed better than the decision tree built during the demonstration.

2. **Building a Gradient Boosting Model**

a. Build a gradient boosting model using the Autotune feature. Add a Gradient Boosting node to the Chapter 3 pipeline, below to the Variable Selection node. Use the Autotune feature. Explore the settings that are made available when Autotune is selected, but keep all properties at their defaults.

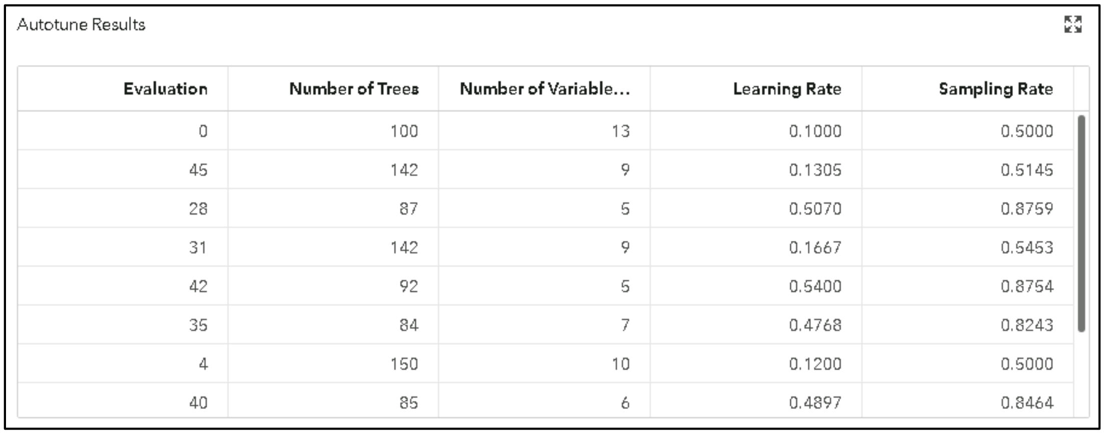
1) On the Starter Template pipeline, right-click the **Variable Selection** node and select **Add below ⇨ Supervised Learning ⇨ Gradient Boosting**.

2) In the properties pane, turn on the **Perform Autotuning** option. The default properties show starting values and ranges that are tried for each property in the Gradient Boosting model.

3) Right-click the **Gradient Boosting** node and select **Run**. This process might take few minutes.

4) When the execution is over, right-click the **Gradient Boosting** node and select **Results**.

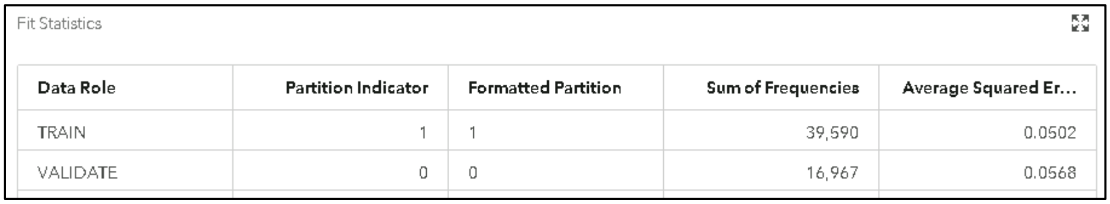
5) Examine the Results window. Maximize the Autotune Results window and notice the different evaluations performed. Restore the Autotune Results window.



Autotune Results

Evaluation	Number of Trees	Number of Variable...	Learning Rate	Sampling Rate
0	100	13	0.1000	0.5000
45	142	9	0.1305	0.5145
28	87	5	0.5070	0.8759
31	142	9	0.1667	0.5453
42	92	5	0.5400	0.8754
35	84	7	0.4768	0.8243
4	150	10	0.1200	0.5000
40	85	6	0.4897	0.8464

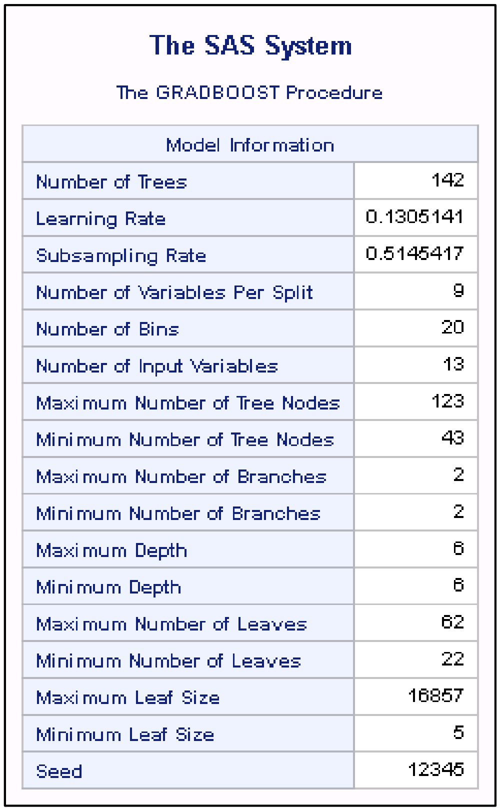
6) Scroll down and observe the Fit Statistics window. The average square error for the Autotune model is 0.0595 on the VALIDATE partition.



Fit Statistics

Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.0502
VALIDATE	0	0	16,967	0.0568

7) Scroll down and maximize the Output window. This output shows the set of parameters selected for the final Gradient Boosting model.



The SAS System

The GRADBOOST Procedure

Model Information	
Number of Trees	142
Learning Rate	0.1305141
Subsampling Rate	0.5145417
Number of Variables Per Split	9
Number of Bins	20
Number of Input Variables	13
Maximum Number of Tree Nodes	123
Minimum Number of Tree Nodes	43
Maximum Number of Branches	2
Minimum Number of Branches	2
Maximum Depth	6
Minimum Depth	6
Maximum Number of Leaves	62
Minimum Number of Leaves	22
Maximum Leaf Size	16857
Minimum Leaf Size	5
Seed	12345

b. What criteria were selected for the champion model?

- The number of trees: **142**
- Number of variables per split: **9**
- Number of bins: **20**
- Maximum number of branches: **2**
- Maximum depth: **6**

c. How does the autotuned Gradient Boosting compare to the other models in the pipeline, particularly to the Gradient Boosting model built during the demonstration? Consider the fit statistic average square error for this comparison.

It performed better than the gradient boosting built during the demonstration.

3. Building a Forest Model

a. Build a forest using the Autotune feature. Add a Forest node to the Chapter 3 pipeline, below the Variable Selection node. Use the Autotune feature. Run few options by changing the range of the parameters search.

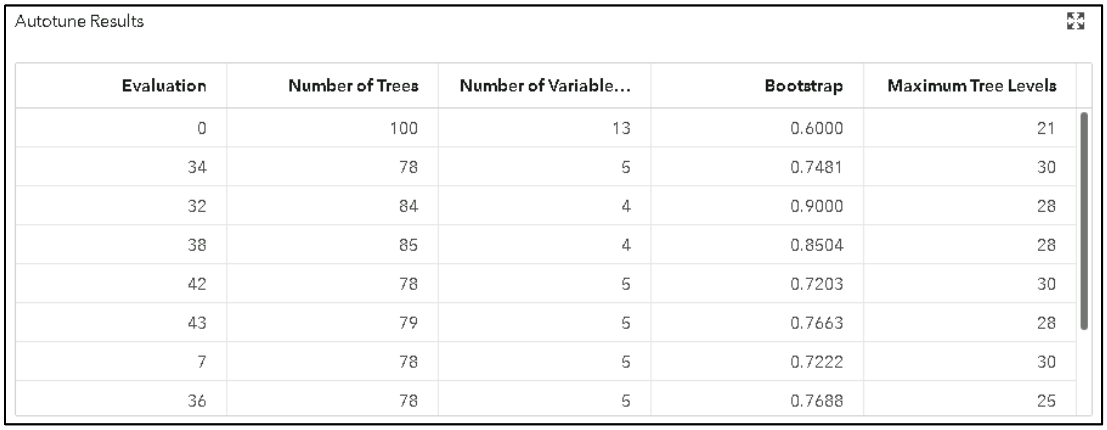
1) On the Starter Template pipeline, right-click the **Variable Selection** node and select **Add below ⇨ Supervised Learning ⇨ Forest**.

2) In the properties pane, turn on the **Perform Autotuning** option. The default properties show starting values and ranges that are tried for each property in the Forest model.

3) Right-click the **Forest** node and select **Run**. This process might take few minutes.

4) When the execution is over, right-click the **Forest** node and select **Results**.

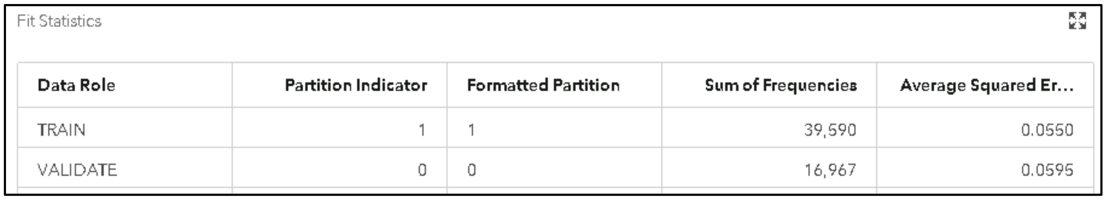
5) Examine the Results window. Maximize the Autotune Results window and notice the different evaluations performed. Restore the Autotune Results window.



Autotune Results

Evaluation	Number of Trees	Number of Variable...	Bootstrap	Maximum Tree Levels
0	100	13	0.6000	21
34	78	5	0.7481	30
32	84	4	0.9000	28
38	85	4	0.8504	28
42	78	5	0.7203	30
43	79	5	0.7663	28
7	78	5	0.7222	30
36	78	5	0.7688	25

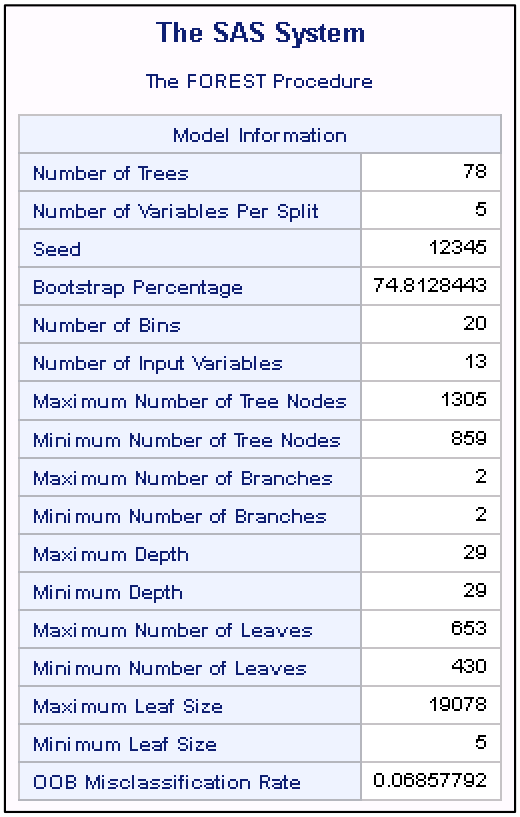
6) Scroll down and observe the Fit Statistics window. The average square error for the Autotune model is 0.0588 on the VALIDATE partition.



Fit Statistics

Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.0550
VALIDATE	0	0	16,967	0.0595

7) Scroll down and maximize the Output window. This output shows the set of parameters selected for the final Forest model.



The SAS System

The FOREST Procedure

Model Information	
Number of Trees	78
Number of Variables Per Split	5
Seed	12345
Bootstrap Percentage	74.8128443
Number of Bins	20
Number of Input Variables	13
Maximum Number of Tree Nodes	1305
Minimum Number of Tree Nodes	859
Maximum Number of Branches	2
Minimum Number of Branches	2
Maximum Depth	29
Minimum Depth	29
Maximum Number of Leaves	653
Minimum Number of Leaves	430
Maximum Leaf Size	19078
Minimum Leaf Size	5
OOB Misclassification Rate	0.06857792

b. What criteria were selected for the champion model?

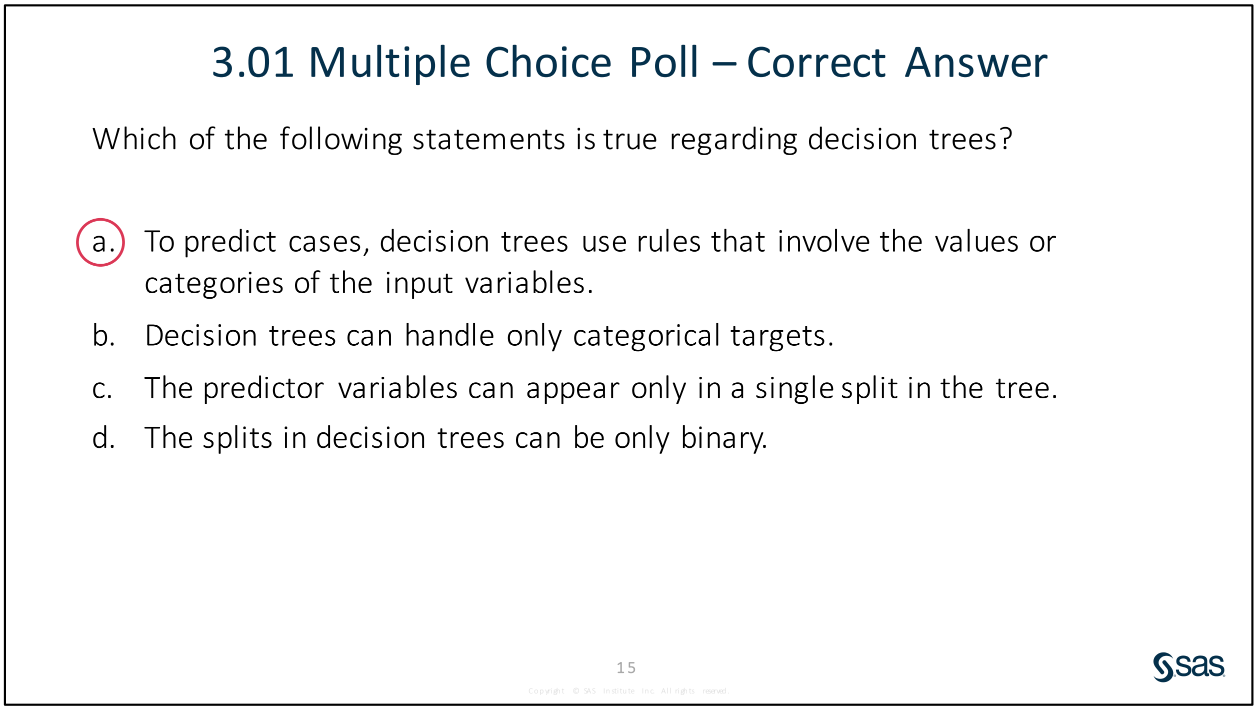
- The number of trees: **78**
- Number of variables per split: **5**
- Number of bins: **20**
- Maximum number of branches: **2**
- Maximum depth: **29**

c. How does the autotuned forest compare to the other models in the pipeline, particularly to the Forest model built during the demonstration? Consider the fit statistic average square error for this comparison.

It performed better than the Forest built during the demonstration.

End of Solutions

Solutions to Student Activities (Polls/Quizzes)



3.01 Multiple Choice Poll – Correct Answer

Which of the following statements is true regarding decision trees?

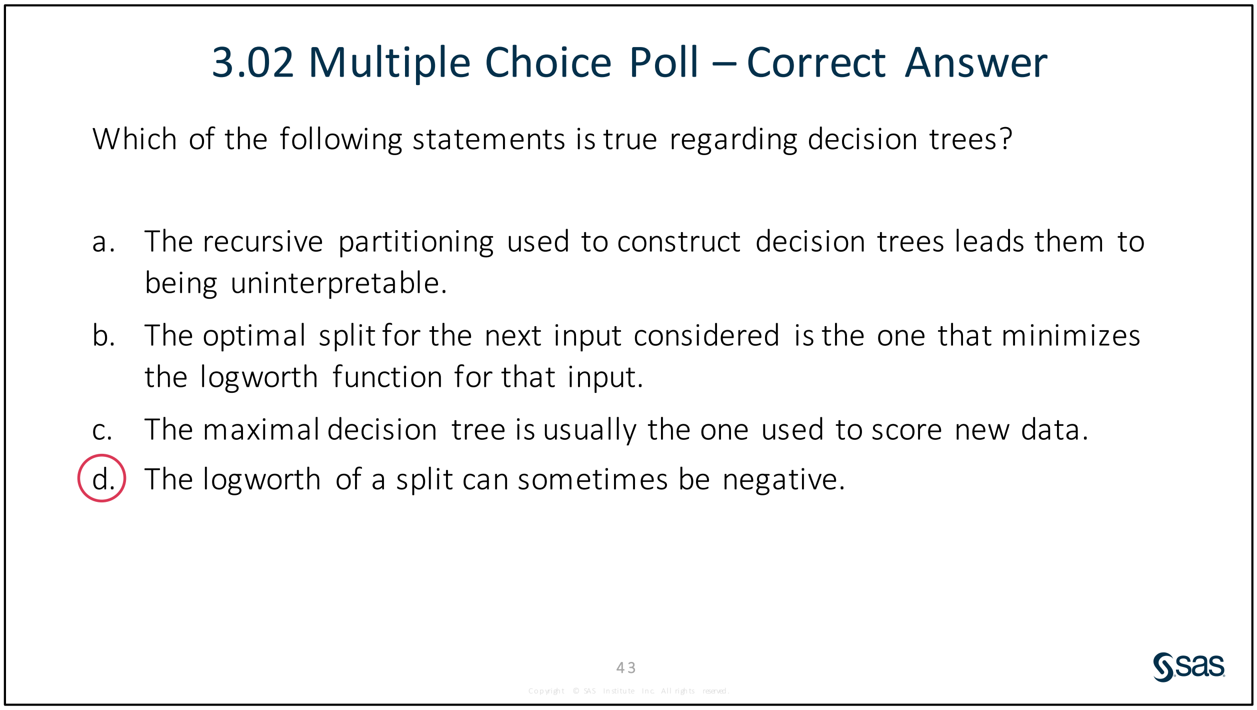
[✓ a.] To predict cases, decision trees use rules that involve the values or categories of the input variables.

b. Decision trees can handle only categorical targets.

c. The predictor variables can appear only in a single split in the tree.

d. The splits in decision trees can be only binary.

15



3.02 Multiple Choice Poll – Correct Answer

Which of the following statements is true regarding decision trees?

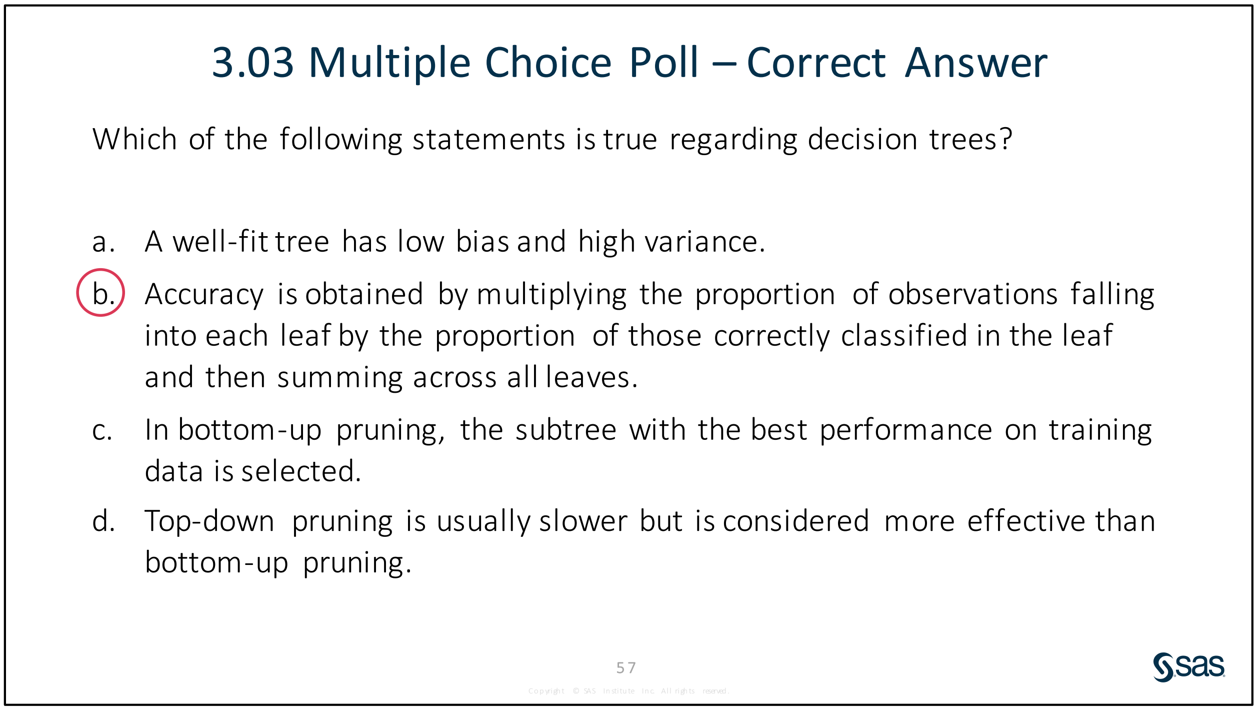
a. The recursive partitioning used to construct decision trees leads them to being uninterpretable.

b. The optimal split for the next input considered is the one that minimizes the logworth function for that input.

c. The maximal decision tree is usually the one used to score new data.

[✓ d.] The logworth of a split can sometimes be negative.

43



3.03 Multiple Choice Poll – Correct Answer

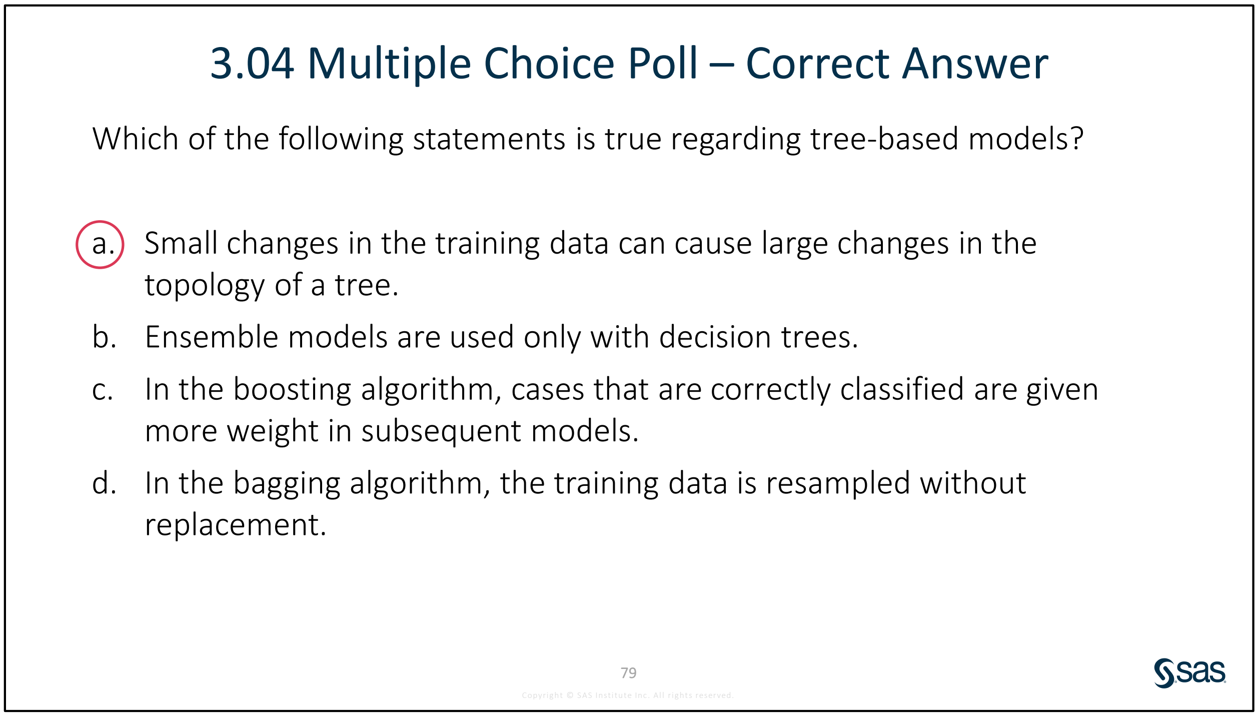
Which of the following statements is true regarding decision trees?

a. A well-fit tree has low bias and high variance.

[✓ b.] Accuracy is obtained by multiplying the proportion of observations falling into each leaf by the proportion of those correctly classified in the leaf and then summing across all leaves.

c. In bottom-up pruning, the subtree with the best performance on training data is selected.

d. Top-down pruning is usually slower but is considered more effective than bottom-up pruning.



3.04 Multiple Choice Poll – Correct Answer

Which of the following statements is true regarding tree-based models?

[✓ a.] Small changes in the training data can cause large changes in the topology of a tree.

b. Ensemble models are used only with decision trees.

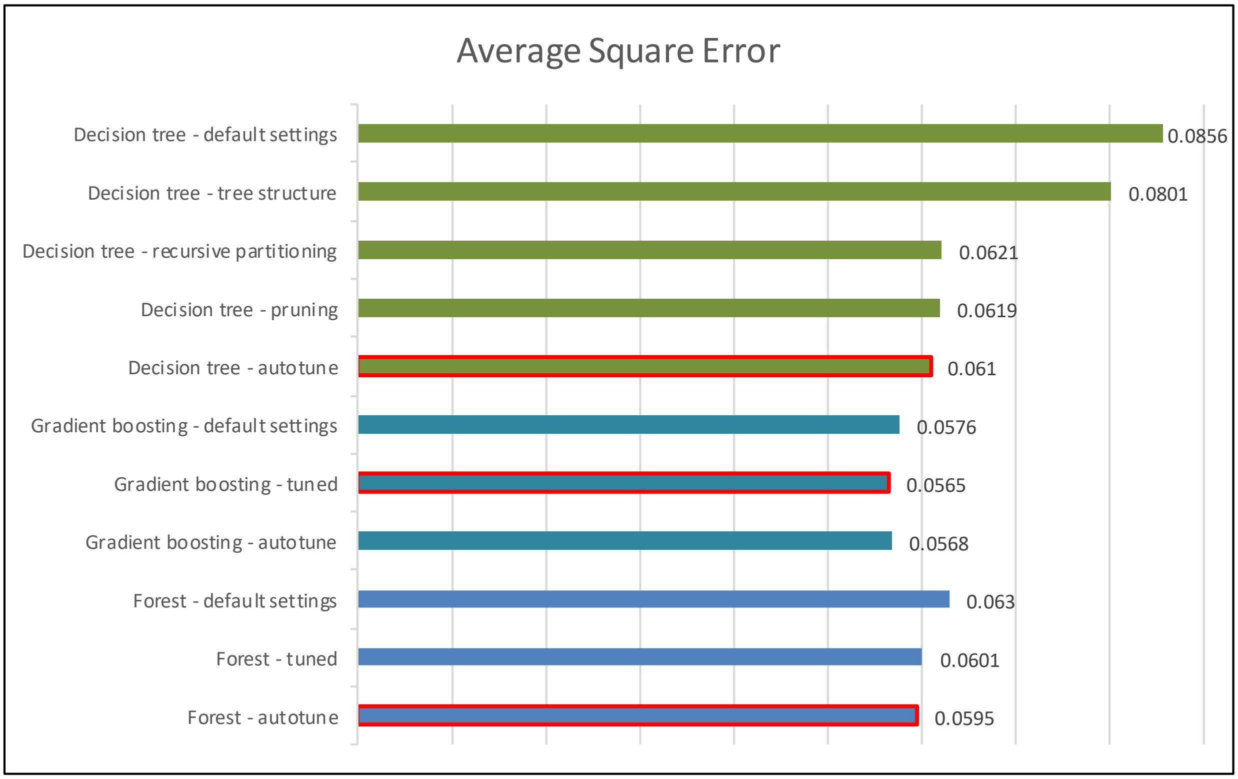
c. In the boosting algorithm, cases that are correctly classified are given more weight in subsequent models.

d. In the bagging algorithm, the training data is resampled without replacement.

Summary of Decision Tree, Forest, and Gradient Boosting Models

The following chart shows the different models' performance for decision tree, gradient boosting, and forest. The decision tree model was improved based on the topics covered in this chapter, such as different tree structures, criteria for recursive partitioning, and pruning approaches. For the decision tree, starting from the default settings, each one of those topics was applied to improve the model's performance. Finally, the decision tree based on the Autotune feature was created.

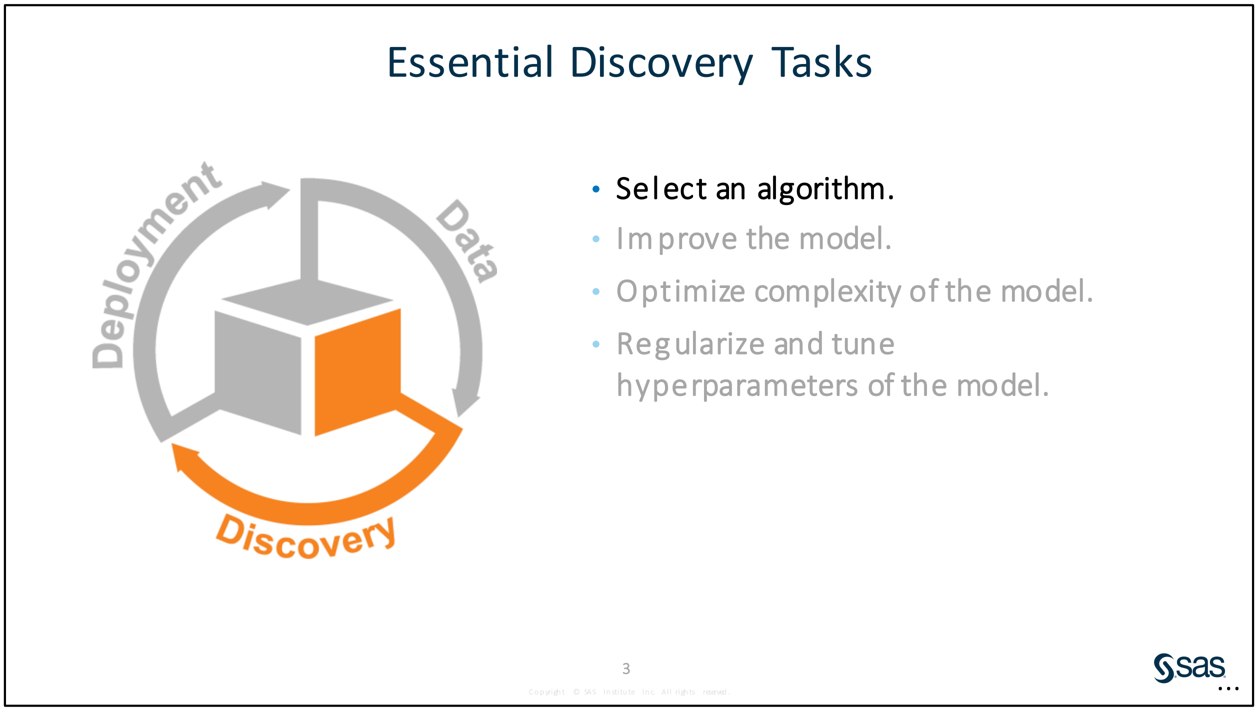
Similarly, for gradient boosting and forest, we started from the model with the default settings and tuned the models to increase performance. At the end, we ran both gradient boosting and forest based on the Autotune feature.



The Autotune feature achieved the best model performance for the decision tree and for the forest. The tuned model built during the demonstration achieved the best model performance for the gradient boosting.

Chapter 4 Neural Networks

4.1 Introduction

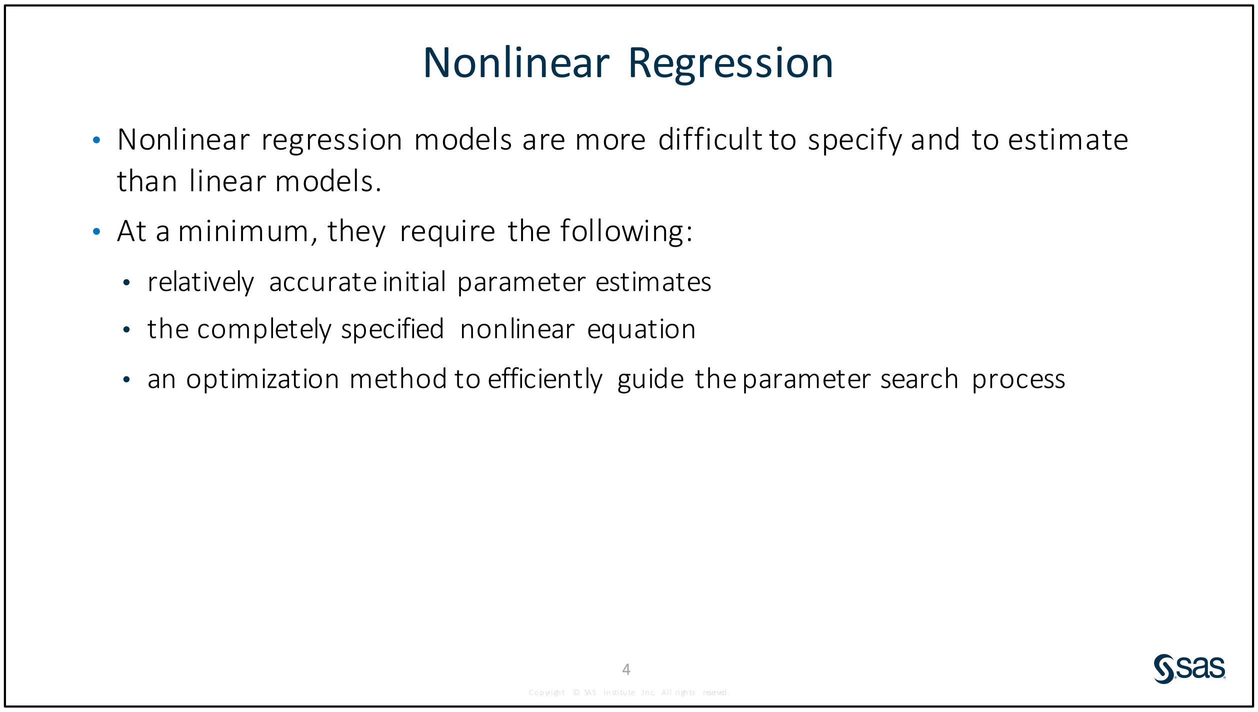


Essential Discovery Tasks

- Select an algorithm.
- Improve the model.
- Optimize complexity of the model.
- Regularize and tune hyperparameters of the model.

3

The discovery phase of the analytic life cycle continues. At this point we have the appropriate data, we have explored and created new features on the data, and we are now ready to use machine learning algorithms to build predictive models or discover patterns in the data. In the previous chapter, we created and refined decision tree and tree-based models. Now we continue the experimentation with different types of approaches by building neural network models.



Nonlinear Regression

- Nonlinear regression models are more difficult to specify and to estimate than linear models.
- At a minimum, they require the following:
  - relatively accurate initial parameter estimates
  - the completely specified nonlinear equation
  - an optimization method to efficiently guide the parameter search process

4

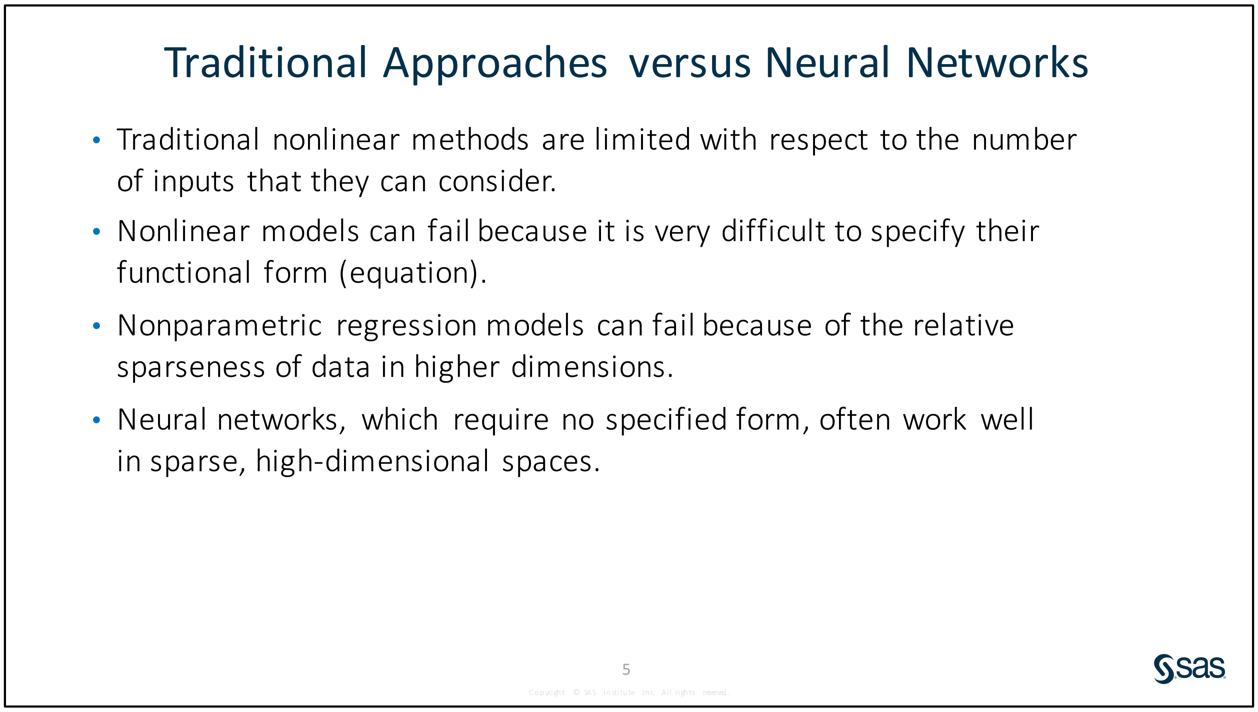
Assuming that the functional form (that is, the equation) defining the nonlinear relationship between *y* and *x* is known *a priori*, and that only the parameters are unknown, then the parameters can be estimated using a technique known as *nonlinear regression* (Seber and Wild 1989).

Note: If the nonlinear equation is ***not*** known, one option is to assume that the input-output relationship takes on some hypothesized functional form. A popular choice is a polynomial. A single input, *x*, polynomial of degree *d* is given by the equation:

$$\hat{y} = w_0 + w_1 x + w_2 x^2 + \ldots + w_d x^d = w_0 + \sum_{k=1}^{d} w_k x^k$$

One reason for the polynomial's popularity is the *Weierstrass approximation theorem*. The theorem asserts that any continuous-valued function on a real interval [a:b], can be approximated arbitrarily closely by a polynomial function. This means that a linear regression model using polynomials of sufficient complexity is actually a universal approximator.

Nonlinear regression models are more difficult to estimate than linear models. Not only must you specify the full nonlinear regression expression to be modeled, an optimization method must be used to efficiently guide the parameter search process. You must also provide initial parameter estimates. The value of these initial parameter estimates is critical. Starting at a bad location in the parameter space results in an inferior solution or, perhaps, even failure to achieve convergence at all.



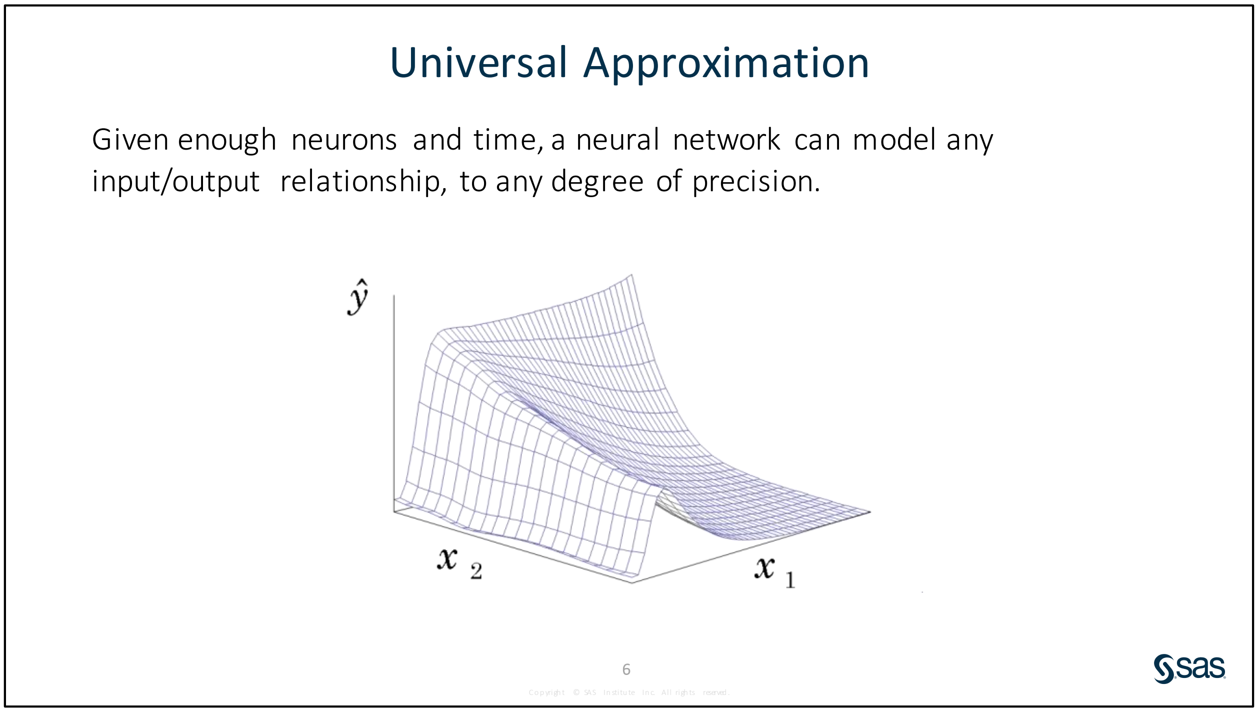
Traditional Approaches versus Neural Networks

- Traditional nonlinear methods are limited with respect to the number of inputs that they can consider.
- Nonlinear models can fail because it is very difficult to specify their functional form (equation).
- Nonparametric regression models can fail because of the relative sparseness of data in higher dimensions.
- Neural networks, which require no specified form, often work well in sparse, high-dimensional spaces.

5

Traditional nonlinear modeling techniques become vastly more difficult as the number of inputs increase. For example, it is uncommon to see parametric nonlinear regression models with more than a few inputs, because a suitable parametric function is usually not derivable. And smoothing splines and other nonparametric regression methods suffer because of the relative sparseness of data in higher dimensions.

In contrast, neural networks are also flexible multivariate function estimators that generally perform well in sparse, high-dimensional spaces. Moreover, it is not necessary to specify their functional form.



Universal Approximation

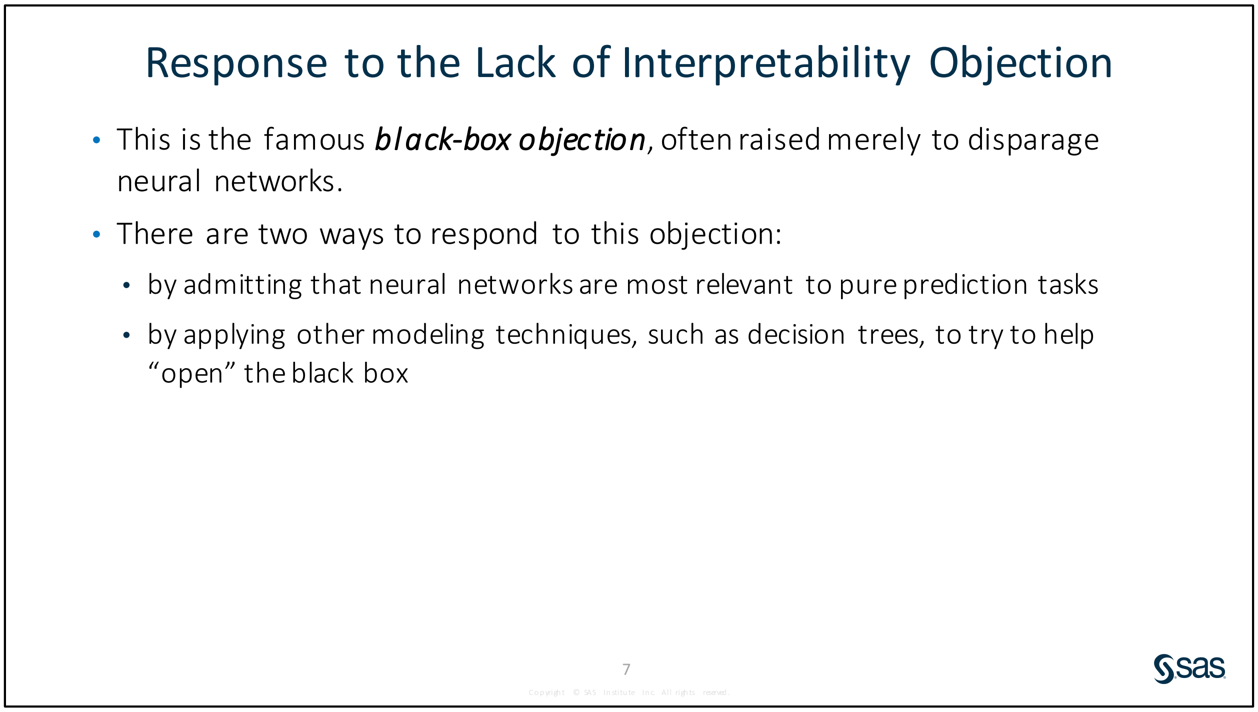
Given enough neurons and time, a neural network can model any input/output relationship, to any degree of precision.

6

Although neural networks are parametric nonlinear regression models, they behave like nonparametric regression (smoothing splines), in that it is not necessary to specify the functional form of the model. This enables construction of models when the relationship between the inputs and outputs is unknown.

The chief benefit of artificial neural networks is their unlimited flexibility. A neural network is a universal approximator, which means that, with a sufficient number of hidden units and enough time, a neural network can model any input-output relationship, no matter how complex.

The question of how many layers and, in particular, how many neurons are needed in each layer for a given modeling task is a very difficult one to answer. This issue is discussed later in the chapter.



Response to the Lack of Interpretability Objection

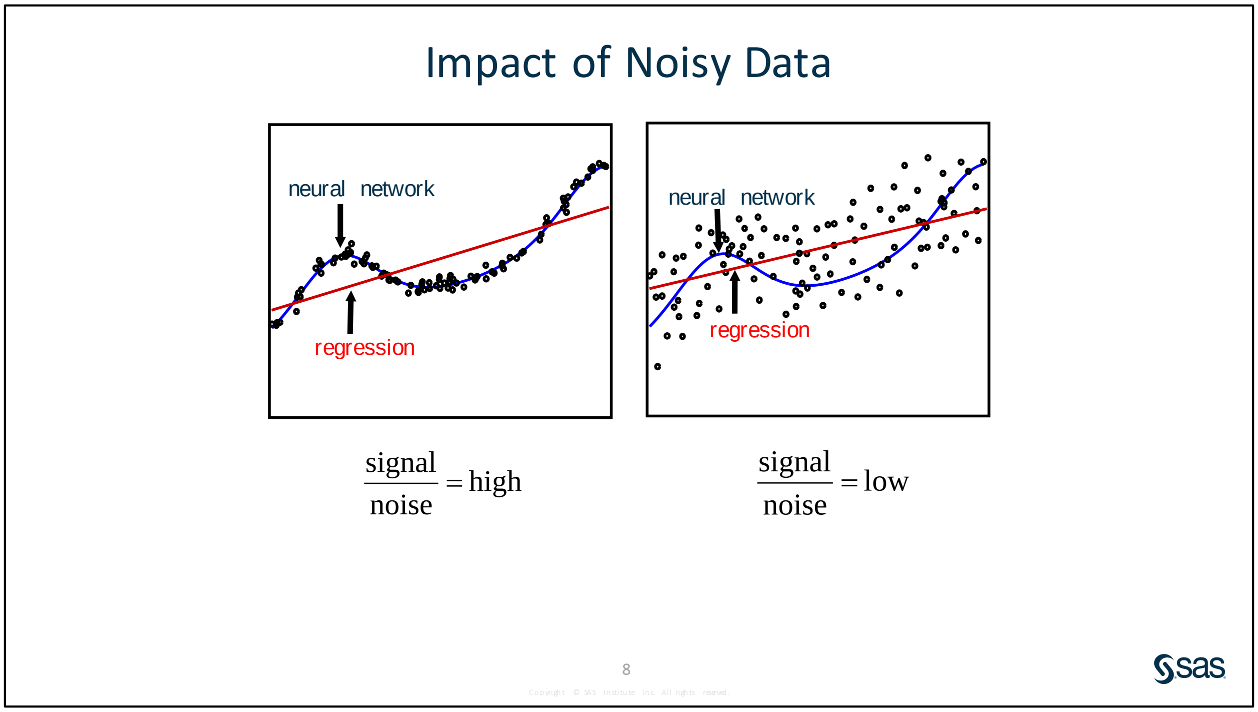
- This is the famous ***black-box objection***, often raised merely to disparage neural networks.
- There are two ways to respond to this objection:
  - by admitting that neural networks are most relevant to pure prediction tasks
  - by applying other modeling techniques, such as decision trees, to try to help “open” the black box

7

This is the famous *black box* objection to neural networks. Suggesting mystery, the term ***black box*** is often used simply to disparage neural networks by implying that they can never be interpreted.

There are two ways to respond to this criticism.

1. The first response is to recognize that, in many tasks, pure prediction is the goal. Understanding how the inputs affect the prediction is of secondary importance. In other applications, the opposite is true: predictive power is a consideration only to the extent that it validates the interpretive power of the model. Neural networks are most appropriate for pure prediction tasks.
2. The second response is to show ways in which the black box has been “opened,” at least partially. Two approaches are direct weight examination through Hinton diagrams and input sensitivity assessment. A particularly interesting method of opening the black box is *decomposition* (Tsukimoto 2000), which contends that the weights in a neural network can be approximated by a set of IF-THEN rules. A related approach is to use a decision tree to interpret the neural network’s predictions.



Impact of Noisy Data

$$\frac{\text{signal}}{\text{noise}} = \text{high} \qquad \frac{\text{signal}}{\text{noise}} = \text{low}$$

Sometimes neural networks do not outperform simpler and easier to implement models like regression. This has led to disenchantment with the more complex neural networks. A possible explanation for this failure was suggested by David Shepard Associates (1999) in *The New Direct Marketing*:

> "…if marketers think they can blindly use [neural networks] without the aid of an experienced statistician or an AI expert, they are making, in our opinion, a very serious mistake …"

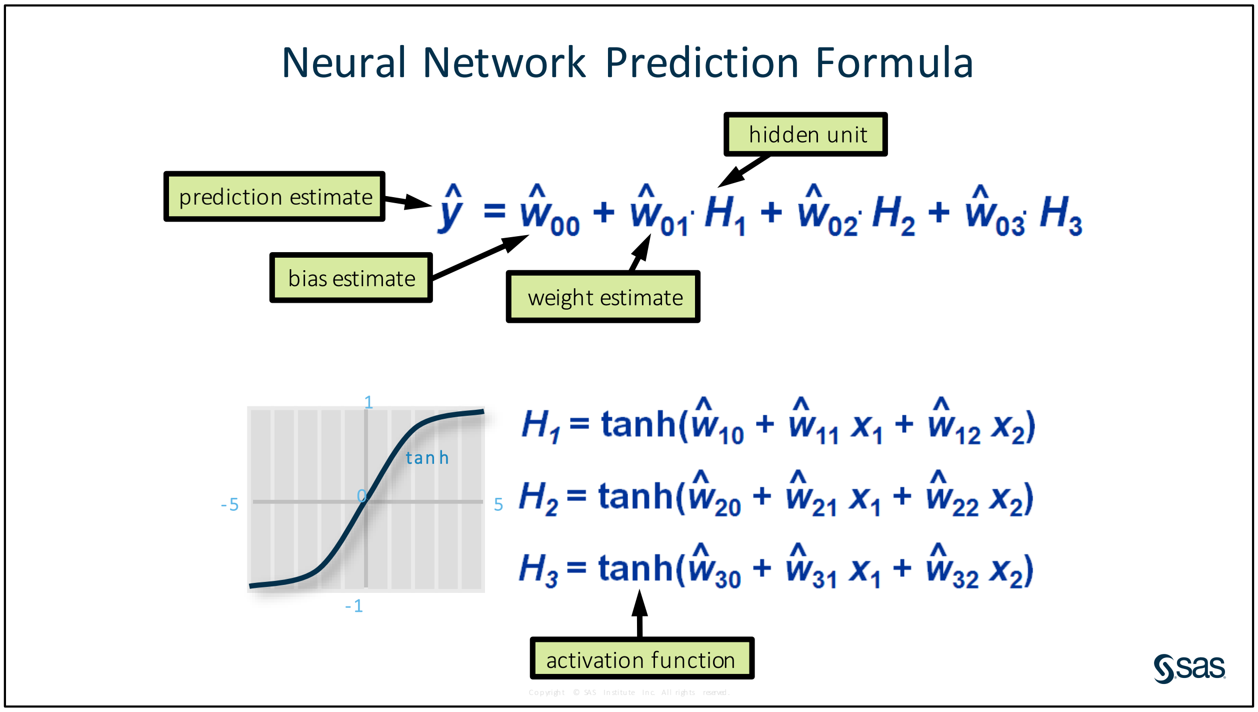
Not only is there a wide array of neural network architectures available today, the staggering number of options within any given architecture makes successful model construction less likely if the modeler is unfamiliar with the theoretical and practical implications of each option.

Another possible explanation for this disenchantment with the relative performance of neural networks in some situations is found in the signal-to-noise ratio.

To illustrate, in the left panel of the above diagram, the nonlinear model that is produced by the nonlinear neural network would clearly produce a superior fit to the data, as compared to the fit produced by the linear regression model. There is a strong pattern (signal) relative to the amount of variation (noise) around the pattern. The signal-to-noise ratio is high.

The situation in the right panel is different: the signal-to-noise ratio is low. The regression and neural network models will likely offer a comparable fit to the data. Therefore, Ockham's razor would imply that, in this case, there would be no advantage to using the more complex neural network.

Note: Ockham's razor states that "entities must not be multiplied beyond necessity." Thus, if competing hypotheses are equal in other respects, Ockham's razor recommends choosing the hypothesis with the fewest postulates. In short, the simplest hypothesis is usually the correct one.



Neural Network Prediction Formula

$$\hat{y} = \hat{w}_{00} + \hat{w}_{01} \cdot H_1 + \hat{w}_{02} \cdot H_2 + \hat{w}_{03} \cdot H_3$$

prediction estimate

bias estimate

weight estimate

hidden unit

$$H_1 = \tanh(\hat{w}_{10} + \hat{w}_{11}\, x_1 + \hat{w}_{12}\, x_2)$$

$$H_2 = \tanh(\hat{w}_{20} + \hat{w}_{21}\, x_1 + \hat{w}_{22}\, x_2)$$

$$H_3 = \tanh(\hat{w}_{30} + \hat{w}_{31}\, x_1 + \hat{w}_{32}\, x_2)$$

activation function

With its exotic sounding name, a Neural Network model (formally, for the models discussed in this course, *multilayer perceptrons*) is often regarded as a mysterious and powerful predictive tool. Perhaps surprisingly, the most typical form of the model is, in fact, a natural extension of a regression model.

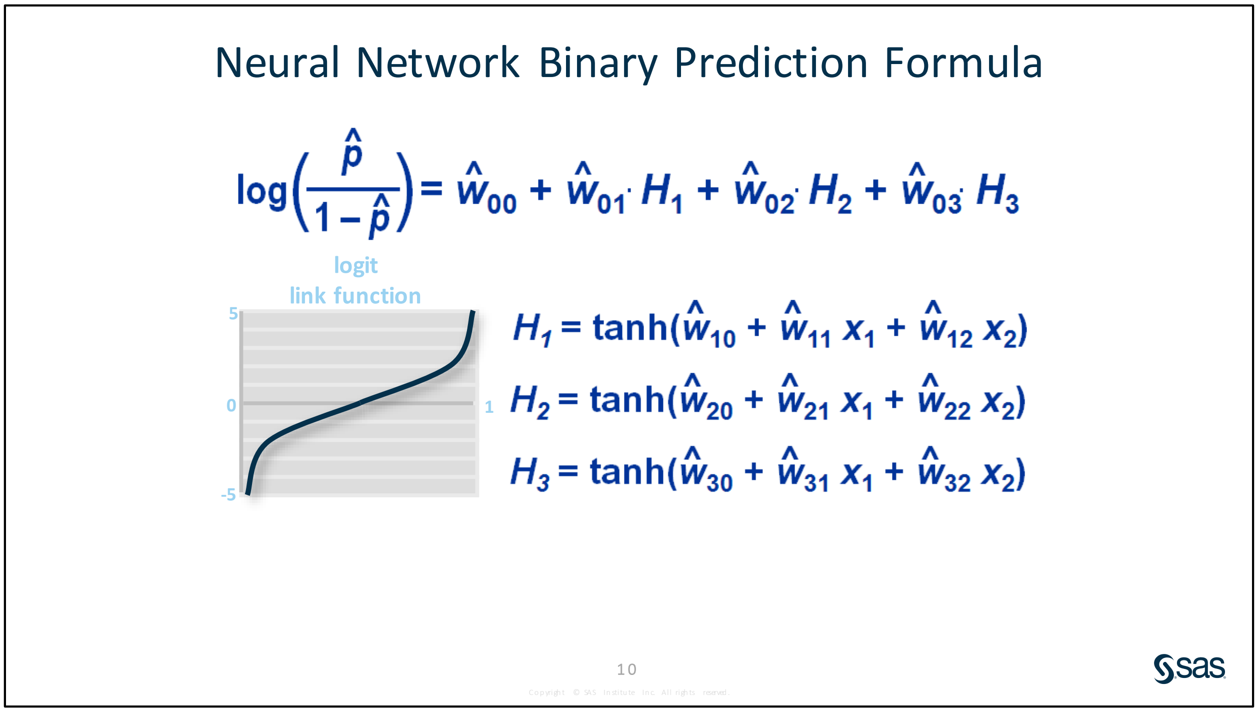
The prediction formula used to predict new cases is similar to a regression's, but with an interesting and flexible addition. This addition enables a properly trained neural network to model virtually any association between input and target variables. However, flexibility comes at a price because the problem of input selection is not easily addressed by a neural network. The inability to select inputs is offset (somewhat) by a complexity optimization method named *stopped training*. Stopped training can reduce the chances of overfitting, even in the presence of redundant and irrelevant inputs.

Like regressions, neural networks predict cases using a mathematical equation involving the values of the input variables.

A neural network can be thought of as a regression model on a set of derived inputs, called *hidden units*. In turn, the hidden units can be thought of as regressions on the original inputs. The hidden unit “regressions” include a default link function (in neural network language, an *activation function*), the hyperbolic tangent. The hyperbolic tangent is a shift and rescaling of the logistic function.

Because of a neural network’s biological roots, its components receive different names from corresponding components of a regression model. Instead of an intercept estimate, a neural network has a *bias* estimate. Instead of parameter estimates, a neural network has *weight estimates*.

What makes neural networks interesting is their ability to approximate virtually any continuous association between the inputs and the target. You simply specify the correct number of hidden units and find reasonable values for the weights. Specifying the correct number of hidden units involves some trial and error. Finding reasonable values for the weights is done by least squares estimation (for interval-valued targets).



Neural Network Binary Prediction Formula

$$\log\left(\frac{\hat{p}}{1-\hat{p}}\right) = \hat{w}_{00} + \hat{w}_{01} \cdot H_1 + \hat{w}_{02} \cdot H_2 + \hat{w}_{03} \cdot H_3$$

logit
link function

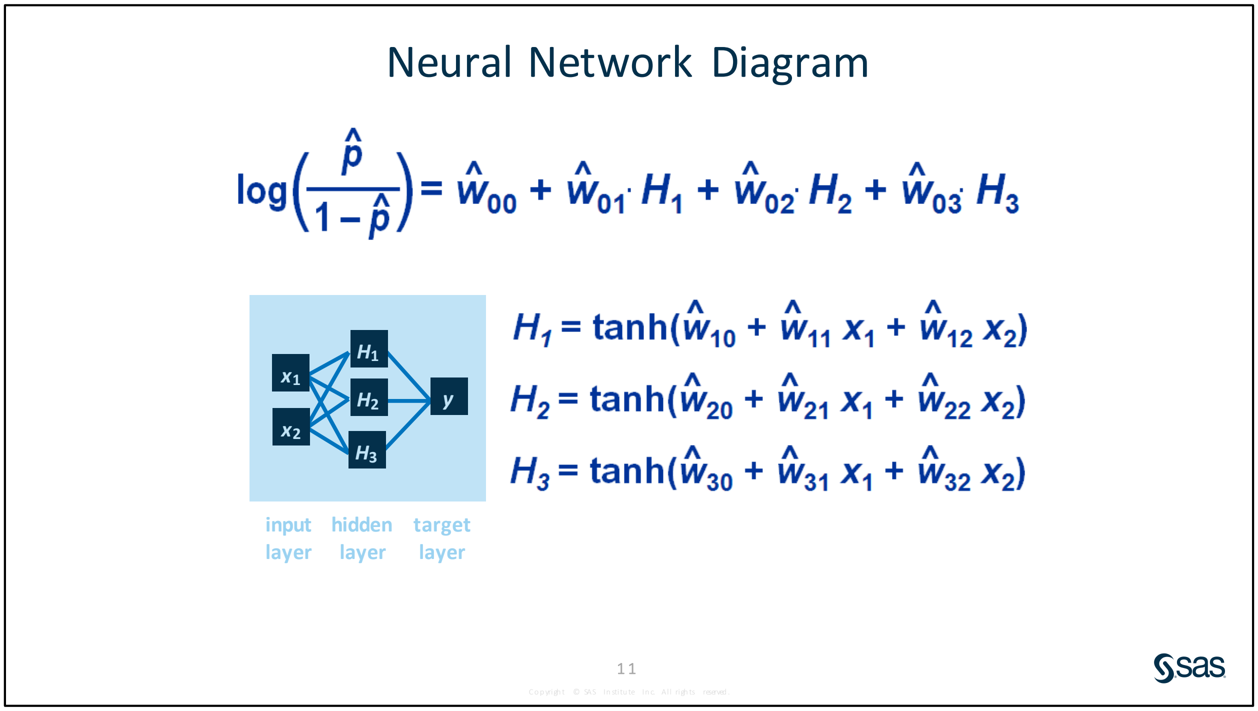
$$H_1 = \tanh(\hat{w}_{10} + \hat{w}_{11}\, x_1 + \hat{w}_{12}\, x_2)$$

$$H_2 = \tanh(\hat{w}_{20} + \hat{w}_{21}\, x_1 + \hat{w}_{22}\, x_2)$$

$$H_3 = \tanh(\hat{w}_{30} + \hat{w}_{31}\, x_1 + \hat{w}_{32}\, x_2)$$

10

When the target variable is binary, as in the demonstration data, the main neural network regression equation receives the same logit link function featured in logistic regression. As with logistic regression, the weight estimation process changes from least squares to maximum likelihood.



Neural Network Diagram

$$\log\left(\frac{\hat{p}}{1-\hat{p}}\right) = \hat{w}_{00} + \hat{w}_{01} \cdot H_1 + \hat{w}_{02} \cdot H_2 + \hat{w}_{03} \cdot H_3$$

x_1
x_2
H_1
H_2
H_3
y

input layer hidden layer target layer

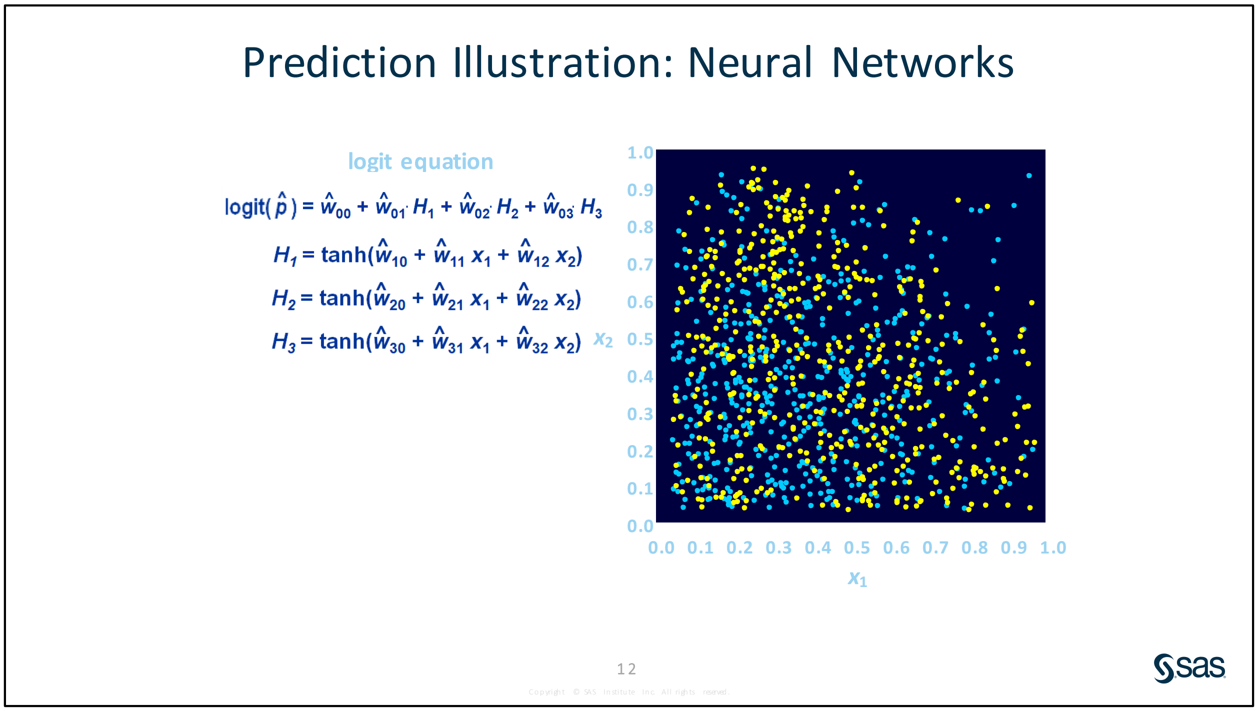
$$H_1 = \tanh(\hat{w}_{10} + \hat{w}_{11}\, x_1 + \hat{w}_{12}\, x_2)$$

$$H_2 = \tanh(\hat{w}_{20} + \hat{w}_{21}\, x_1 + \hat{w}_{22}\, x_2)$$

$$H_3 = \tanh(\hat{w}_{30} + \hat{w}_{31}\, x_1 + \hat{w}_{32}\, x_2)$$

11

Multilayer perceptron models were originally inspired by neurophysiology and the interconnections between neurons, and they are often represented by a network diagram instead of an equation. The basic model form arranges neurons in layers. The first layer, called the *input layer*, connects to a layer of neurons called a *hidden layer*, which, in turn, connects to a final layer called the *target* or *output layer*. Each element in the diagram has a counterpart in the network equation. The blocks in the diagram correspond to inputs, hidden units, and target variables. The block interconnections correspond to the network equation weights.



Prediction Illustration: Neural Networks

logit equation

$$\text{logit}(\hat{p}) = \hat{w}_{00} + \hat{w}_{01} \cdot H_1 + \hat{w}_{02} \cdot H_2 + \hat{w}_{03} \cdot H_3$$

$$H_1 = \tanh(\hat{w}_{10} + \hat{w}_{11}\, x_1 + \hat{w}_{12}\, x_2)$$

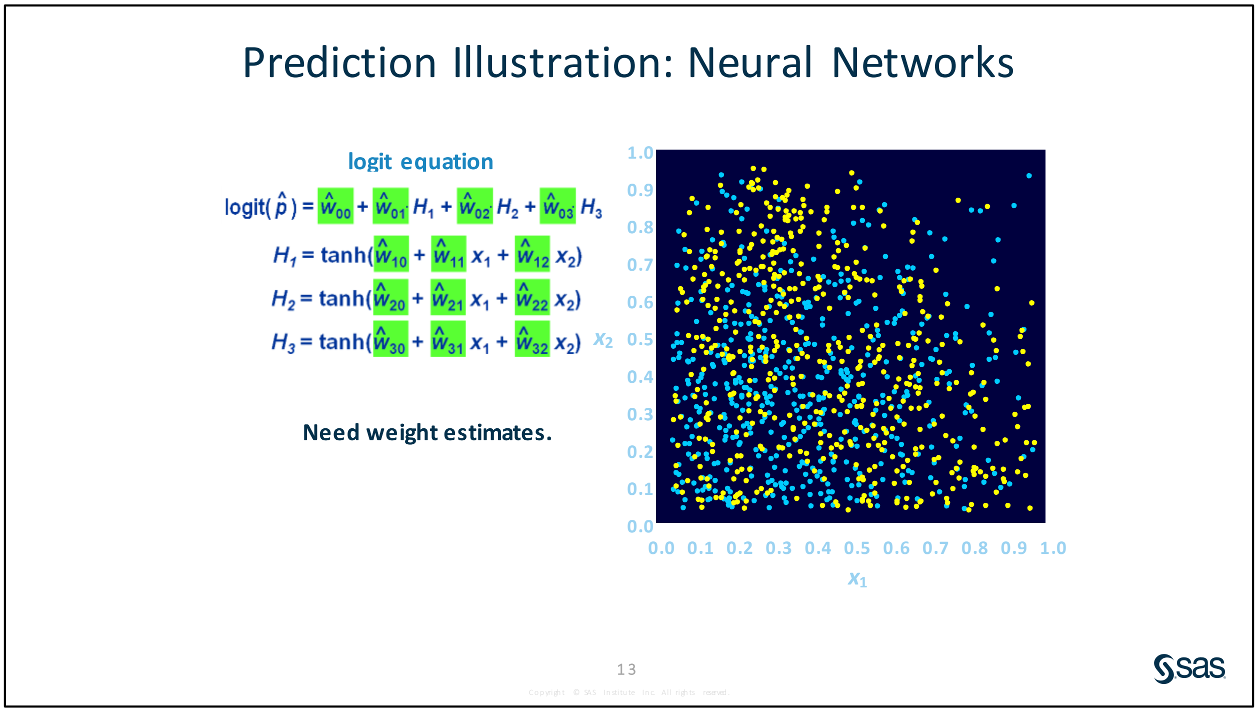
$$H_2 = \tanh(\hat{w}_{20} + \hat{w}_{21}\, x_1 + \hat{w}_{22}\, x_2)$$

$$H_3 = \tanh(\hat{w}_{30} + \hat{w}_{31}\, x_1 + \hat{w}_{32}\, x_2)$$

12

To demonstrate the properties of a neural network model, consider again the two-color prediction problem. As always, the goal is to predict the target color based on the location in the unit square.

As with regressions, the predictions can be decisions, rankings, or estimates. The logit equation produces a ranking or logit score.



Prediction Illustration: Neural Networks

logit equation

$$\text{logit}(\hat{p}) = \hat{w}_{00} + \hat{w}_{01} H_1 + \hat{w}_{02} H_2 + \hat{w}_{03} H_3$$

$$H_1 = \tanh(\hat{w}_{10} + \hat{w}_{11} x_1 + \hat{w}_{12} x_2)$$

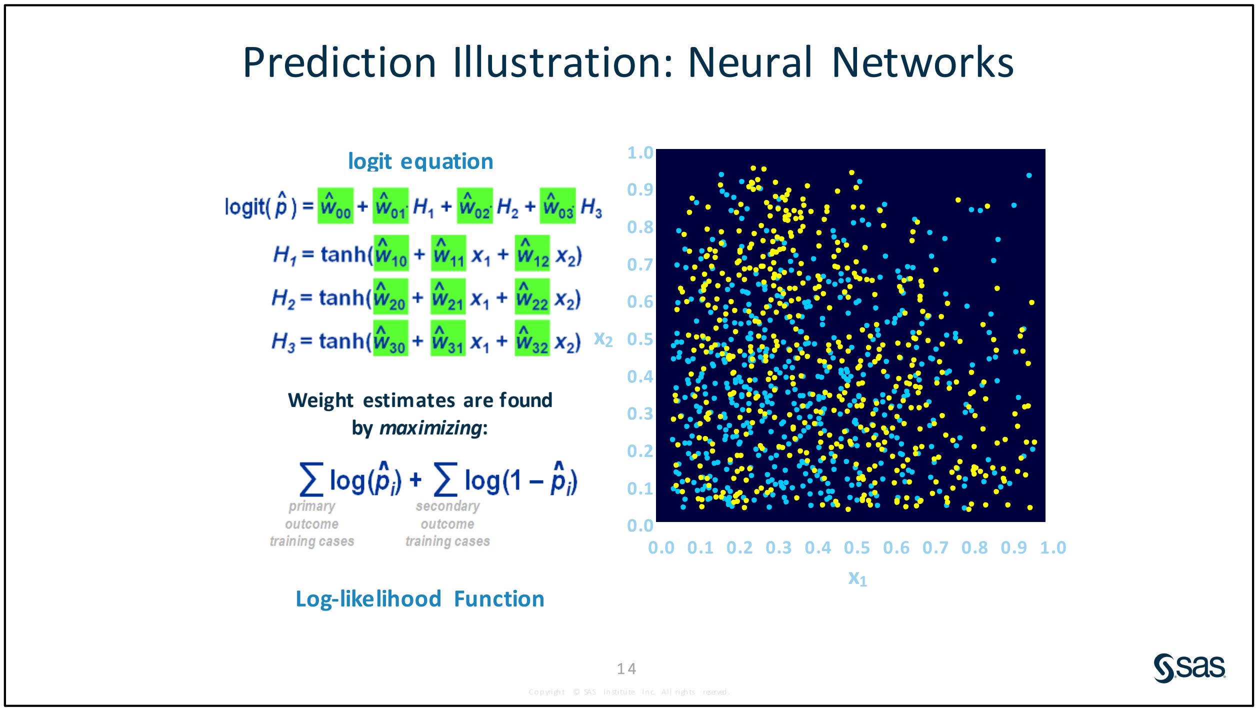
$$H_2 = \tanh(\hat{w}_{20} + \hat{w}_{21} x_1 + \hat{w}_{22} x_2)$$

$$H_3 = \tanh(\hat{w}_{30} + \hat{w}_{31} x_1 + \hat{w}_{32} x_2)$$

Need weight estimates.

1.0
0.9
0.8
0.7
0.6
x_2 0.5
0.4
0.3
0.2
0.1
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
x_1

As with a regression model, the primary task is to obtain parameter, or in the neural network case, weight estimates that result in accurate predictions. However, a major difference between a neural network and a regression model is the number of values to be estimated and the complicated relationship between the weights.



Prediction Illustration: Neural Networks

logit equation

$$\text{logit}(\hat{p}) = \hat{w}_{00} + \hat{w}_{01} H_1 + \hat{w}_{02} H_2 + \hat{w}_{03} H_3$$

$$H_1 = \tanh(\hat{w}_{10} + \hat{w}_{11} x_1 + \hat{w}_{12} x_2)$$

$$H_2 = \tanh(\hat{w}_{20} + \hat{w}_{21} x_1 + \hat{w}_{22} x_2)$$

$$H_3 = \tanh(\hat{w}_{30} + \hat{w}_{31} x_1 + \hat{w}_{32} x_2)$$

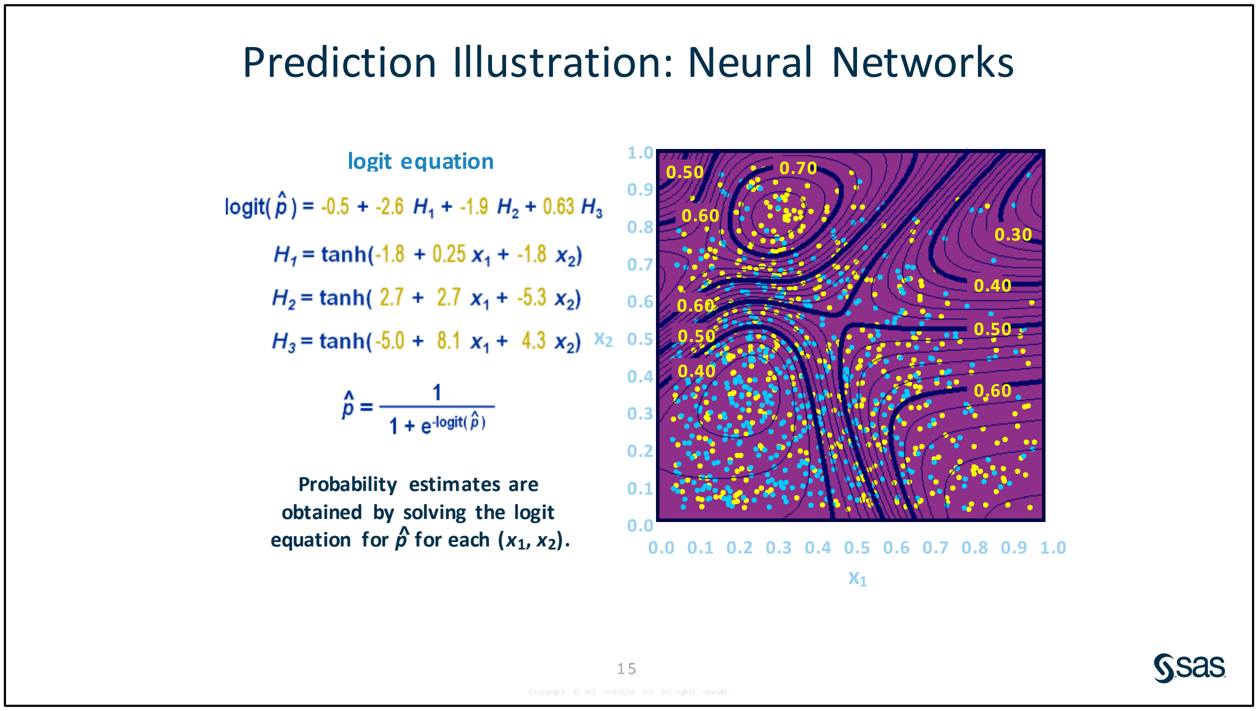
Weight estimates are found by *maximizing*:

$$\underbrace{\sum \log(\hat{p}_i)}_{\substack{\text{primary}\\\text{outcome}\\\text{training cases}}} + \underbrace{\sum \log(1 - \hat{p}_i)}_{\substack{\text{secondary}\\\text{outcome}\\\text{training cases}}}$$

Log-likelihood Function

For a binary target, the weight estimation process is driven by an attempt to maximize the log-likelihood function. Unfortunately, in the case of a Neural Network model, the maximization process is complicated by local optima as well as a tendency to create overfit models. A technique illustrated in the next section (usually) overcomes these difficulties and produces a reasonable model.

Note: By default, the Neural Network task scales all input variables prior to the weight estimation step so that they have a midrange of zero, a minimum of -1, and a maximum of 1. This default standardization method is known as *Midrange*. Other options for this property are standard deviation and no standardization. These settings are controlled by the Standardization Method property, which is found on the Options tab.



Prediction Illustration: Neural Networks

logit equation

$$\text{logit}(\hat{p}) = -0.5 + -2.6\ H_1 + -1.9\ H_2 + 0.63\ H_3$$

$$H_1 = \tanh(-1.8 + 0.25\ x_1 + -1.8\ x_2)$$

$$H_2 = \tanh(\ 2.7 + \ 2.7\ x_1 + -5.3\ x_2)$$

$$H_3 = \tanh(-5.0 + \ 8.1\ x_1 + \ 4.3\ x_2)$$

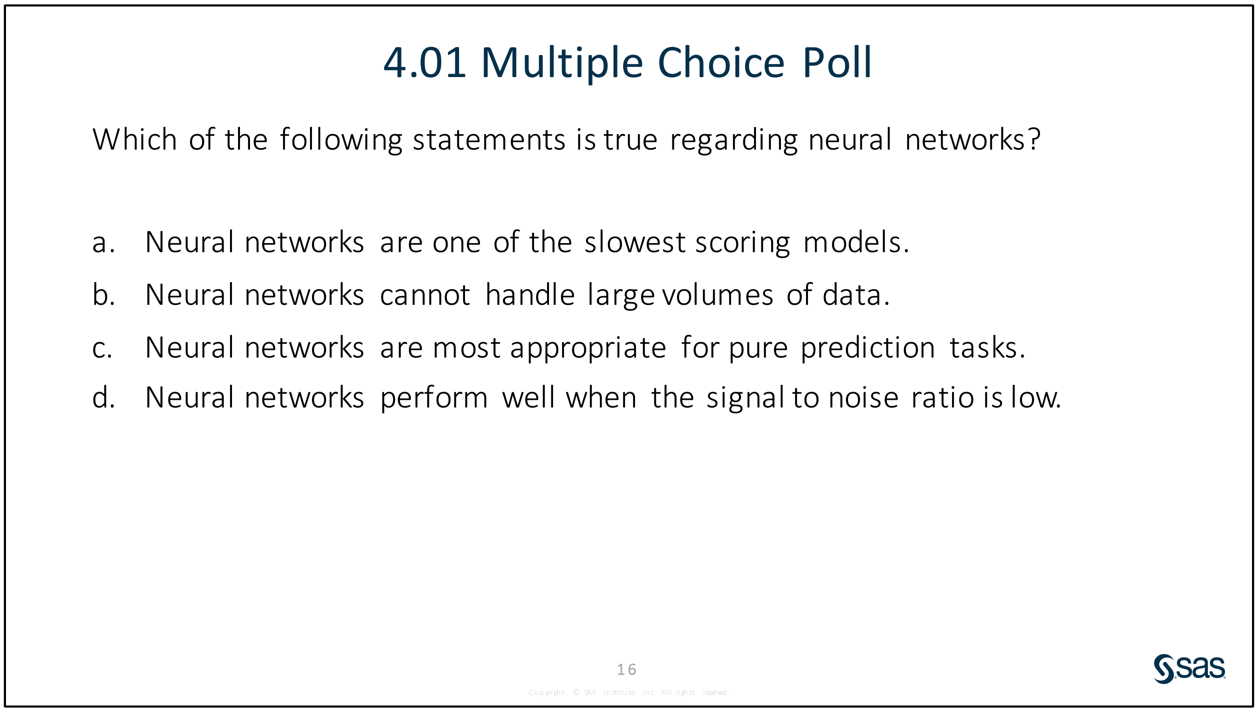
$$\hat{p} = \frac{1}{1 + e^{-\text{logit}(\hat{p})}}$$

Probability estimates are obtained by solving the logit equation for $\hat{p}$ for each (x_1, x_2).

15

Even a relatively simple neural network with three hidden units permits elaborate associations between the inputs and the target. Although the model might be simple, explanation of the model is decidedly not. This lack of explicability is frequently cited as a major disadvantage of a neural network. Of course, complex input or target associations are difficult to explain no matter what technique is used to model them. Neural networks should not be faulted, assuming that they correctly modeled this association.

After the prediction formula is established, obtaining a prediction is strictly a matter of plugging the input measurements into the hidden unit expressions. In the same way as with regression models, you can obtain a prediction estimate using the logistic function.



4.01 Multiple Choice Poll

Which of the following statements is true regarding neural networks?

a. Neural networks are one of the slowest scoring models.

b. Neural networks cannot handle large volumes of data.

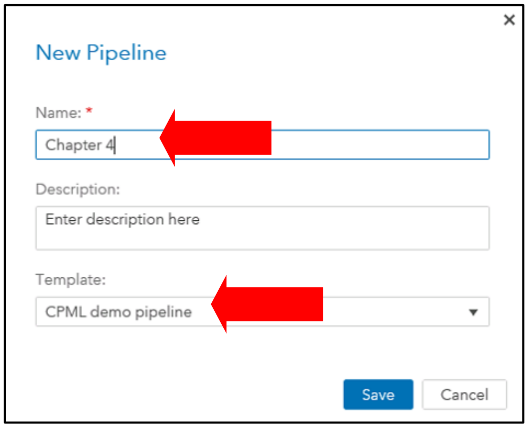
c. Neural networks are most appropriate for pure prediction tasks.

d. Neural networks perform well when the signal to noise ratio is low.

Building a Neural Network Model with Default Settings

In this demonstration, you create a new pipeline using the CPML demo pipeline and add a Neural Network node to it. You build the Neural Network model using the default settings of the node.

1. Click the plus sign (**+**) next to the Chapter 3 pipeline tab to add a new pipeline.
2. In the New Pipeline window, enter **Chapter 4** in the **Name** field, access the menu under the **Template** property, and select **CPML demo pipeline**.



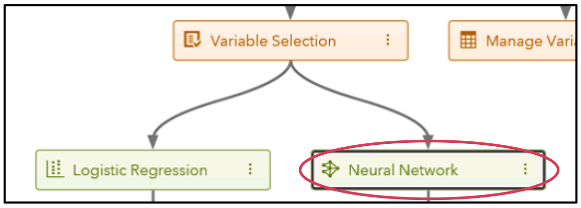
New Pipeline

Name: * Chapter 4

Description: Enter description here

Template: CPML demo pipeline

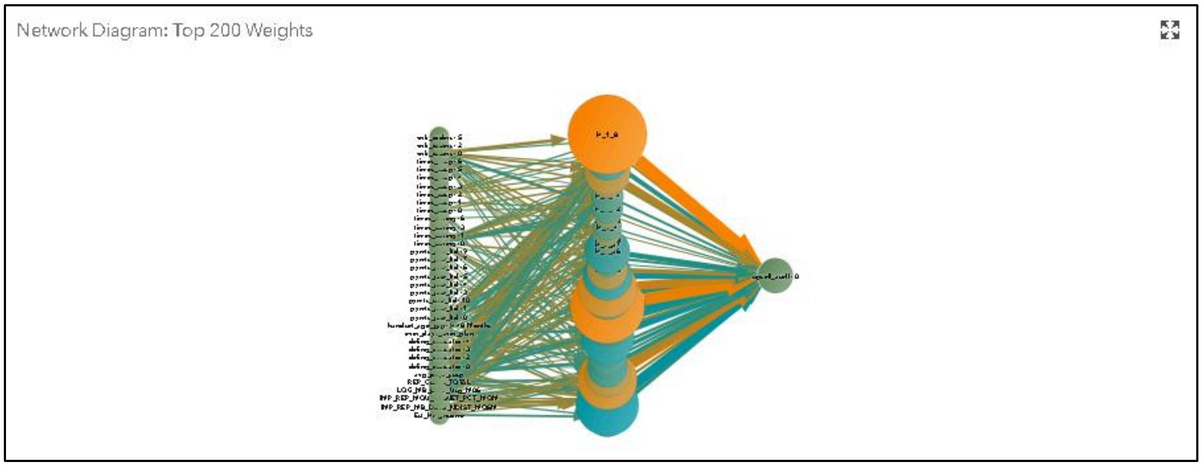
3. Click **Save**.
4. In the Chapter 4 pipeline, right-click the **Variable Selection** node and select **Add below** ⇨ **Supervised Learning** ⇨ **Neural Network**.



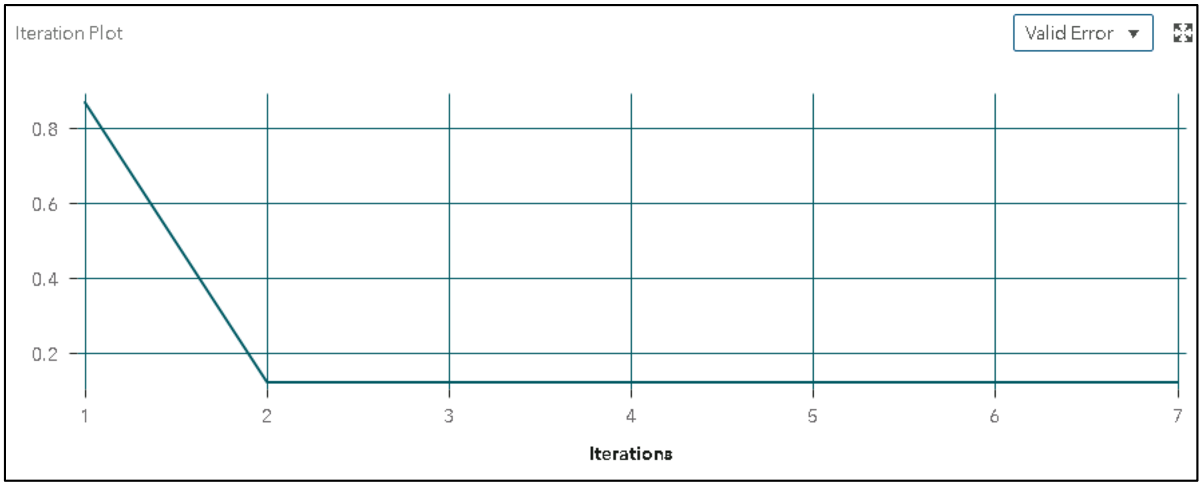
5. Keep all properties for the Neural Network at their defaults.
6. Run the Neural Network node.

7. Open the results for the Neural Network node.

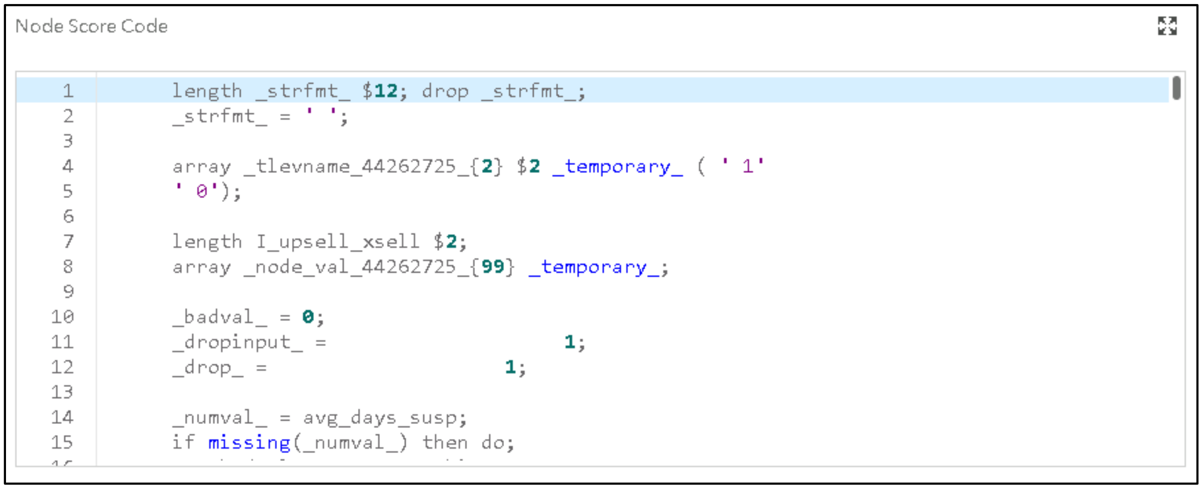
   There are several charts and plots to help you evaluate the model's performance. The first plot is the Network Diagram, which presents the final neural network structure for this model, including the hidden layer and the hidden units.



   The Iteration plot shows the model's performance based on the valid error throughout the training process when new iterations are added to achieve the final model.

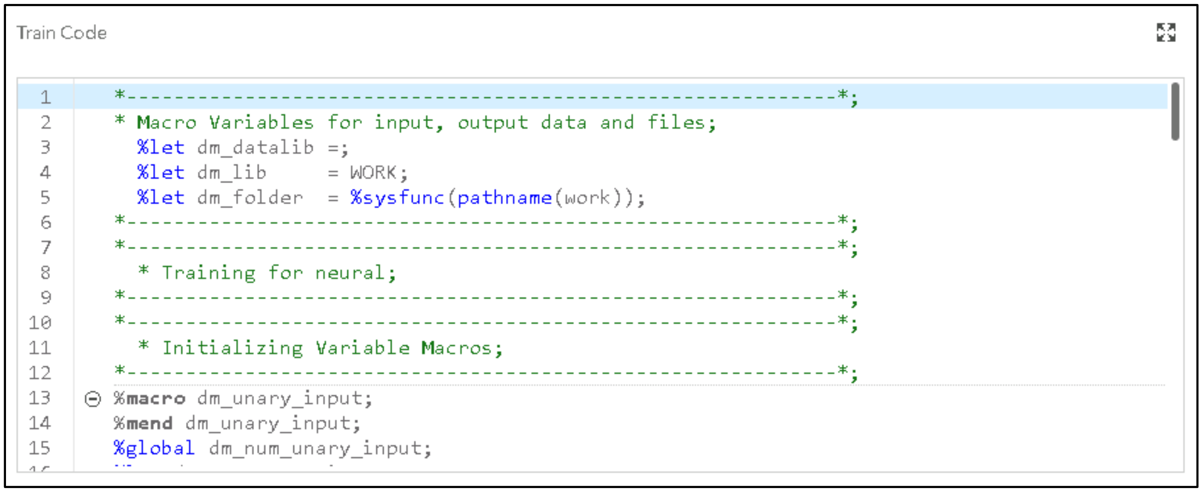


The Node Score Code window shows the final score code that can be deployed in production.



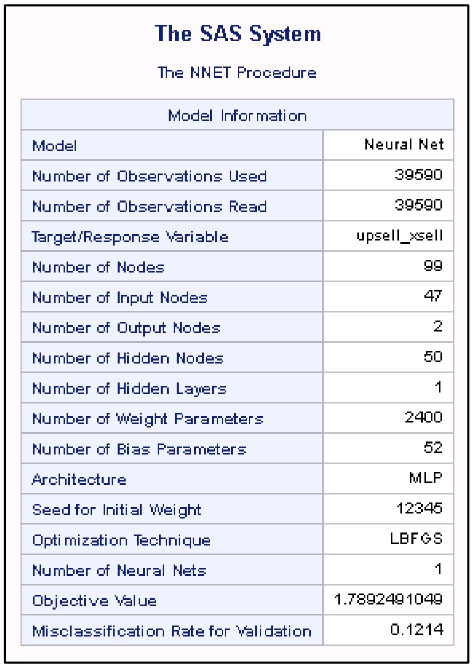
```
      length _strfmt_ $12; drop _strfmt_;
      _strfmt_ = ' ';

      array _tlevname_44262725_{2} $2 _temporary_ ( ' 1'
      ' 0');

      length I_upsell_xsell $2;
      array _node_val_44262725_{99} _temporary_;

      _badval_ = 0;
      _dropinput_ =                      1;
      _drop_ =                      1;

      _numval_ = avg_days_susp;
      if missing(_numval_) then do;
```

Similarly, the Train Code window shows the train code that can be used to train the model based on different data sets or in different platforms.



```
*--------------------------------------------------------------*;
* Macro Variables for input, output data and files;
  %let dm_datalib =;
  %let dm_lib     = WORK;
  %let dm_folder  = %sysfunc(pathname(work));
*--------------------------------------------------------------*;
*--------------------------------------------------------------*;
  * Training for neural;
*--------------------------------------------------------------*;
*--------------------------------------------------------------*;
  * Initializing Variable Macros;
*--------------------------------------------------------------*;
%macro dm_unary_input;
%mend dm_unary_input;
%global dm_num_unary_input;
```

Finally, the Output window shows the final neural network model parameters, the iteration history, and the optimization process.

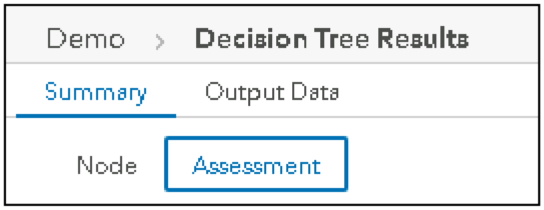


The SAS System

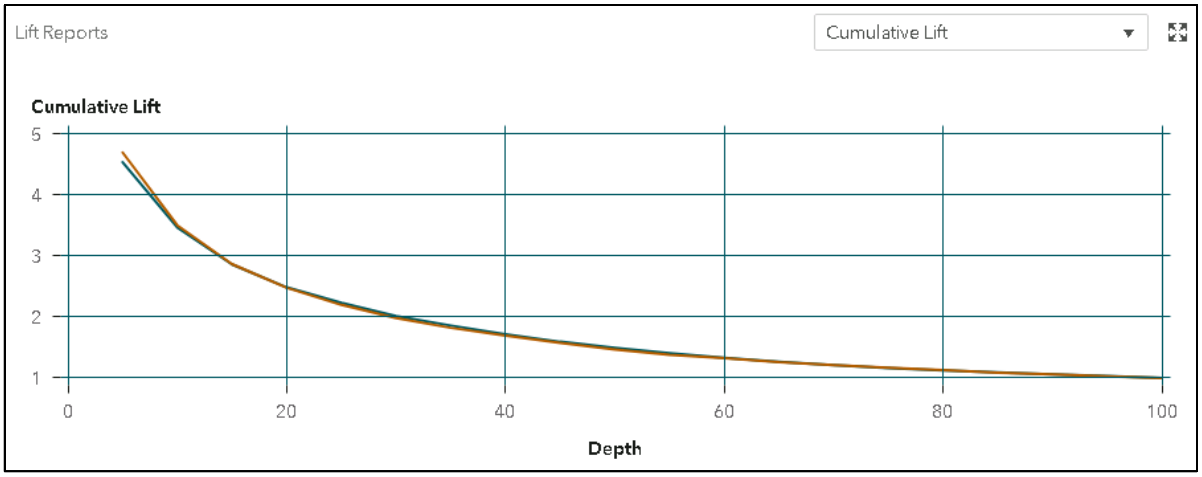
The NNET Procedure

Model Information	
Model	Neural Net
Number of Observations Used	39590
Number of Observations Read	39590
Target/Response Variable	upsell_xsell
Number of Nodes	99
Number of Input Nodes	47
Number of Output Nodes	2
Number of Hidden Nodes	50
Number of Hidden Layers	1
Number of Weight Parameters	2400
Number of Bias Parameters	52
Architecture	MLP
Seed for Initial Weight	12345
Optimization Technique	LBFGS
Number of Neural Nets	1
Objective Value	1.7892491049
Misclassification Rate for Validation	0.1214

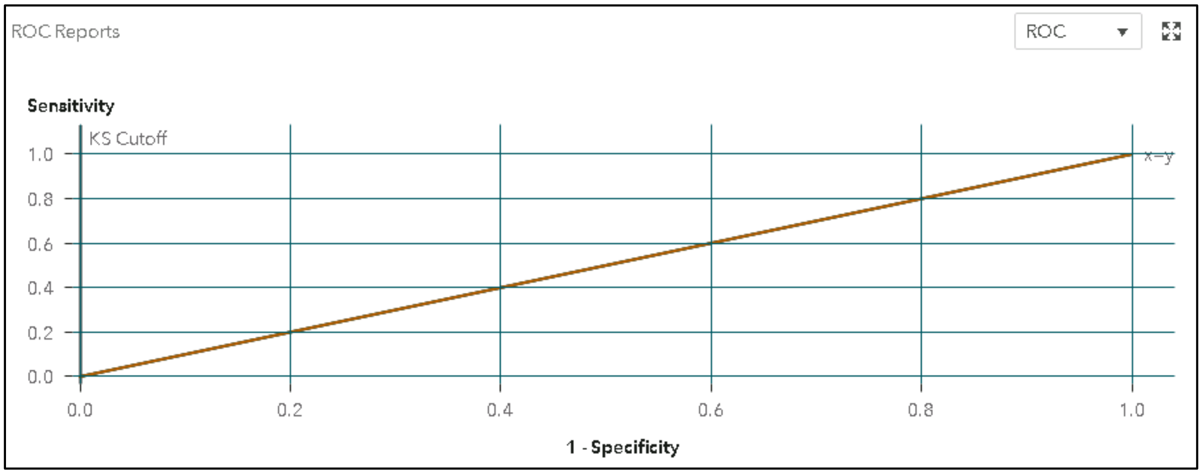
8. Click the **Assessment** tab.



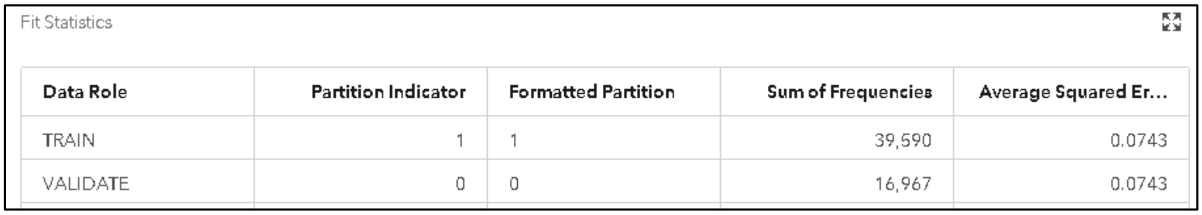
The first chart is the Cumulative Lift, showing the model's performance ordered by the percentage of the population. This chart is very useful for selecting the model based on a particular target of the customer base. It shows how much better the model is than the random events.



For a binary target, you also have the ROC curve, which shows the model's performance considering the true positive rate and the false positive rate. It is good to foresee the performance on a specific business events, when all positive cases are selected. It shows that the model's performance based on the positive cases were predicted right and the positive cases were predicted wrong. ROC is very useful for deployment.



Finally, you have the Fit Statistics output, which shows the model's performance based on some assessment measures, such as average square error.



Fit Statistics

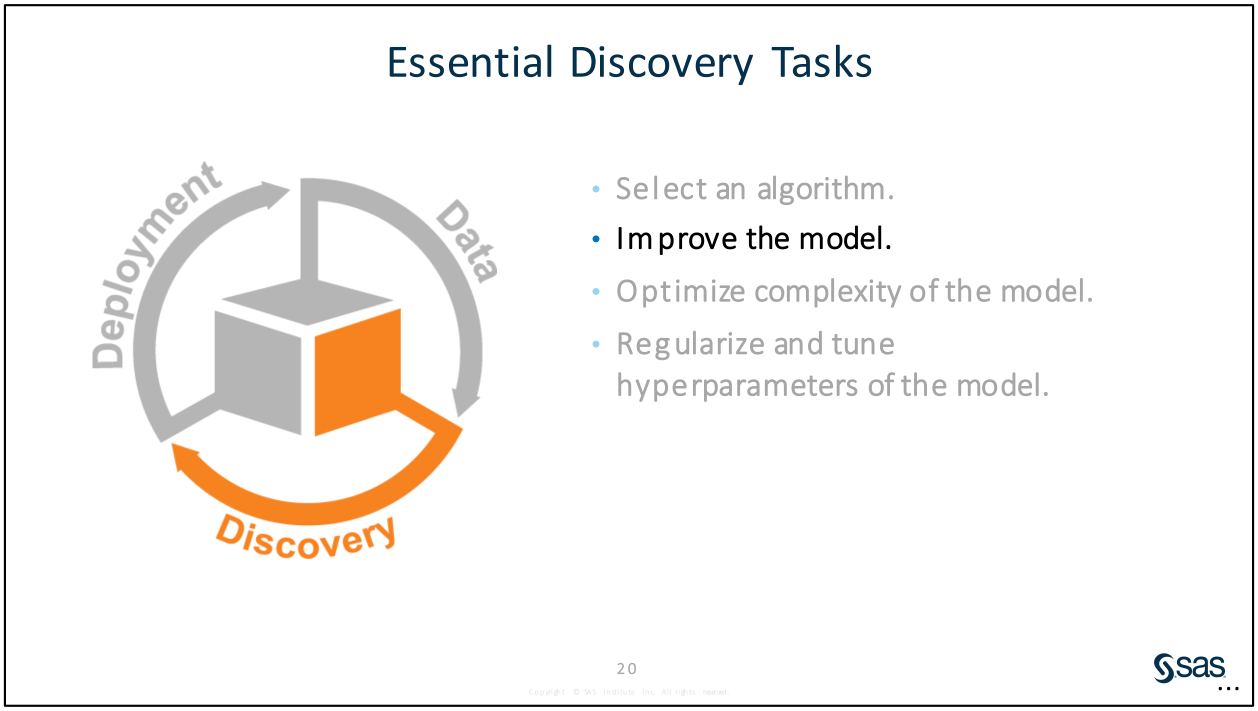
Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.0743
VALIDATE	0	0	16,967	0.0743

The Fit Statistics shows an average square error of 0.0743 on the VALIDATE partition.

9. Close the Results window.

End of Demonstration

4.2 Network Architecture

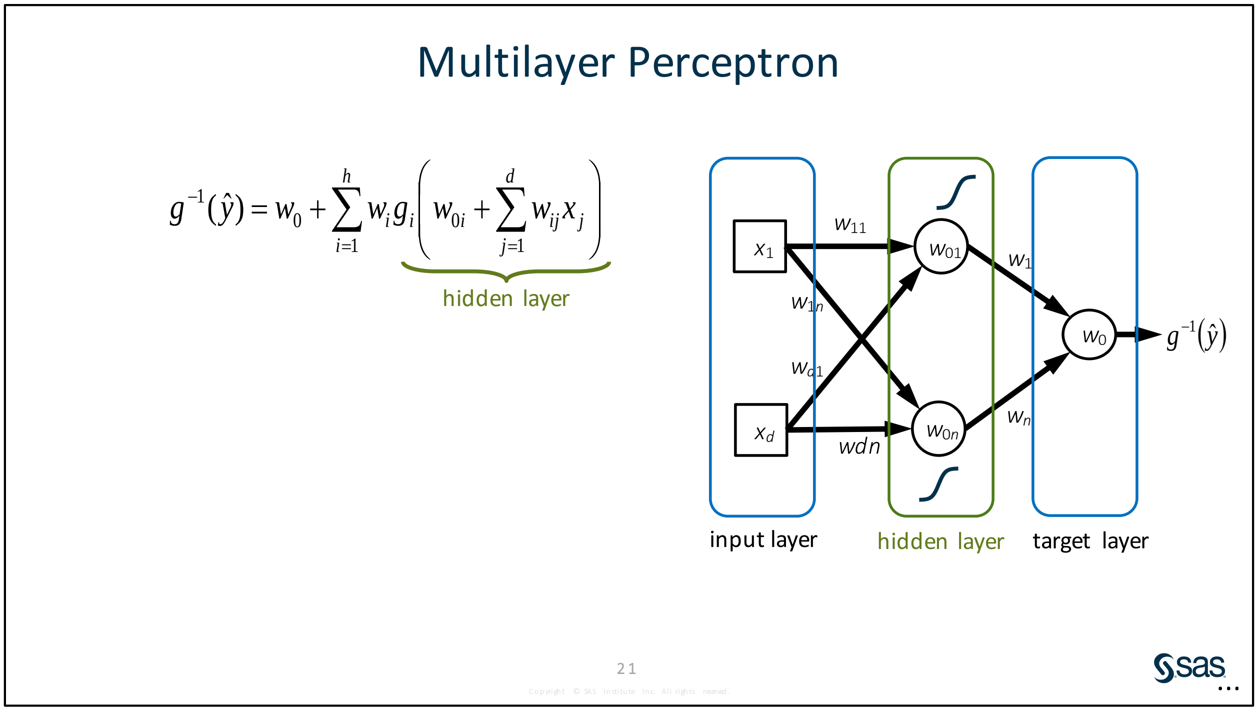


Essential Discovery Tasks

- Select an algorithm.
- Improve the model.
- Optimize complexity of the model.
- Regularize and tune hyperparameters of the model.

Deployment
Data
Discovery
20

One of the methods for improving neural network models are changing their architectures by adding hidden layers and adding hidden units, all of which make the models more flexible.



Multilayer Perceptron

$$g^{-1}(\hat{y}) = w_0 + \sum_{i=1}^{h} w_i g_i \underbrace{\left(w_{0i} + \sum_{j=1}^{d} w_{ij} x_j \right)}_{\text{hidden layer}}$$

There are three layers in the basic multilayer perceptron (MLP) neural network:

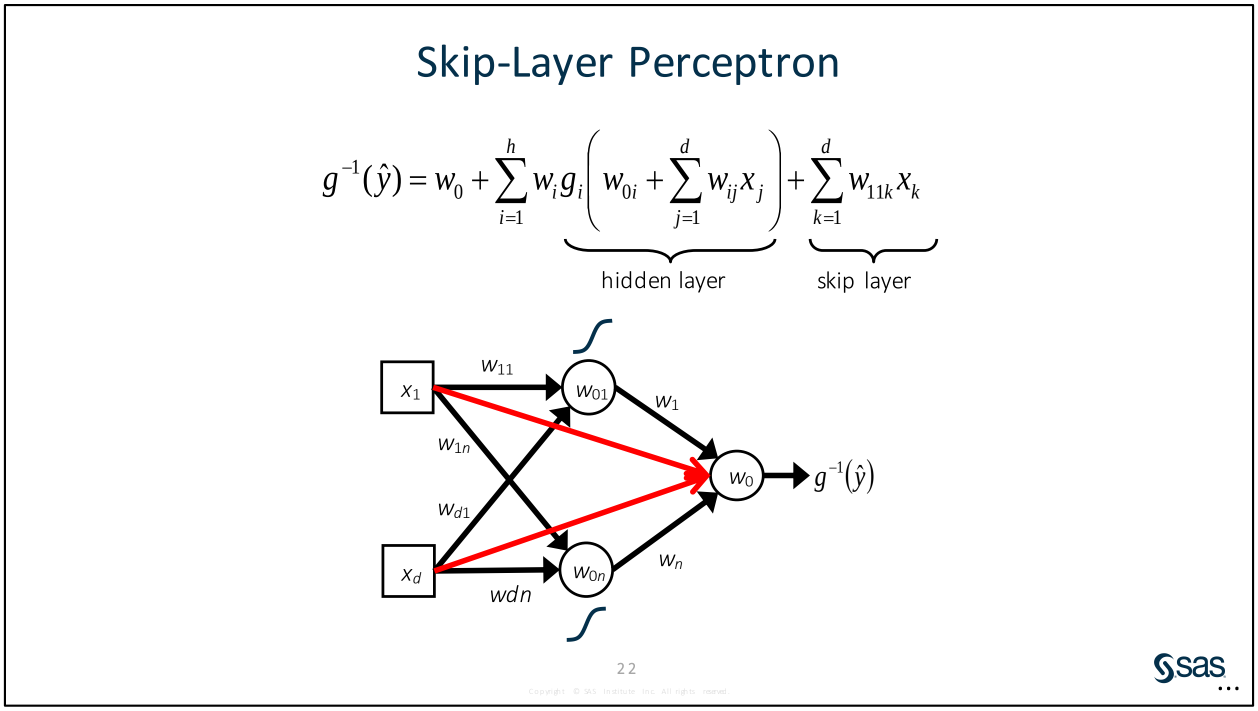
1. An *input layer* contains a neuron/unit for each input variable. The input layer neurons have no adjustable parameters (weights). They simply pass the positive or negative input to the next layer.
2. A *hidden layer* has hidden units that (by default) perform a sigmoidal transformation of the weighted and summed input activations.
3. An *output layer* shapes and combines the nonlinear hidden layer activation values.

A single hidden-layer multilayer perceptron constructs a limited extent region, or *bump*, of large values surrounded by smaller values (Principe et al. 2000). For example, the intersection of the hyper-planes created by a hidden layer consisting of three hidden units forms a triangle-shaped bump.

The hidden and output layers ***must*** be connected by a nonlinear function in order to act as separate layers. Otherwise, the multilayer perceptron collapses into a linear perceptron. More formally, if matrix **A** is the set of weights that transforms input matrix **X** into the hidden layer output values, and matrix **B** is the set of weights that transforms the hidden unit output into the final estimates **Y**, then the linearly connected multilayer network can be represented as **Y**=**B**[**A**(**X**)]. However, if a single-layer weight matrix **C**=**BA** is created, exactly the same output can be obtained from the single-layer network: **Y**=**C**(**X**).

The number of parameters in an MLP with *k* interval inputs grows quickly with the number of hidden units, *h,* considered. The number of parameters is given by the equation $h(k+1)+h+1=h(k+2)+1$.

Note: The "number of parameters" equations in this chapter assume that the inputs are interval or ratio level. Each nominal or ordinal input increases *k* by the number of classes in the variable, minus 1.



Skip-Layer Perceptron

$$g^{-1}(\hat{y}) = w_0 + \sum_{i=1}^{h} w_i \underbrace{g_i\left(w_{0i} + \sum_{j=1}^{d} w_{ij}x_j\right)}_{\text{hidden layer}} + \underbrace{\sum_{k=1}^{d} w_{11k}x_k}_{\text{skip layer}}$$

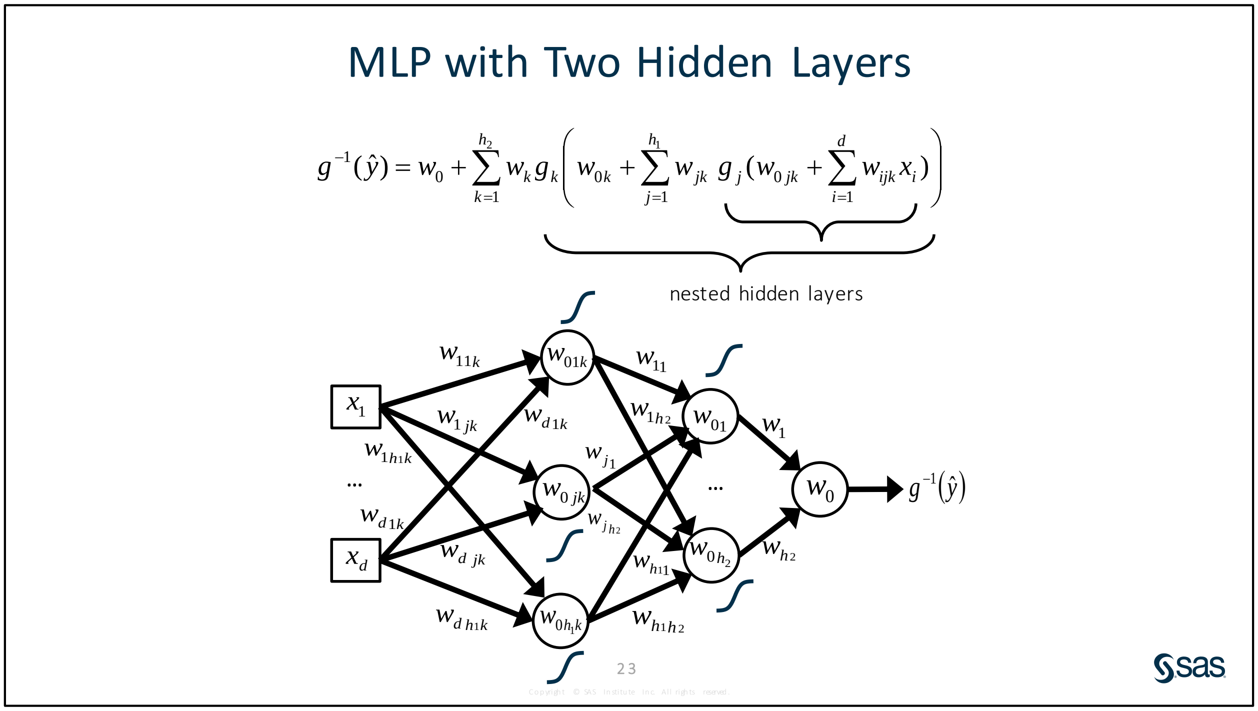
22

By adding direct connections from the input to output layers, bypassing the hidden layer, it is possible to combine the linear and nonlinear neural network paradigms. The result is known as a *skip-layer network*.

Because a multilayer perceptron is already a universal approximator, in general there is little to be gained by adding direct connections. Adding direct connections does ***not*** make it any more universal. However, a multilayer perceptron is inherently stationary (Leisch et al. 1999). This means that it will tend to perform poorly when applied to nonstationary data. In this case, adding direct connections can help.

The number of parameters in a skip-layer network with *k* inputs and *h* hidden units is $h(k+2)+k+1$.

In SAS Visual Data Mining and Machine Learning, skip-layer perceptrons are constructed when the property **Allow direct connections between input and target neurons**, located on the Options tab, is selected.



MLP with Two Hidden Layers

$$g^{-1}(\hat{y}) = w_0 + \sum_{k=1}^{h_2} w_k g_k \left(\underbrace{w_{0k} + \sum_{j=1}^{h_1} w_{jk}\, g_j \underbrace{(w_{0jk} + \sum_{i=1}^{d} w_{ijk} x_i)}} _{\text{nested hidden layers}} \right)$$

23

As Sarle (1997) writes in his Neural Network FAQ, "If you have only one input, there seems to be no advantage to using more than one hidden layer. But things get much more complicated when there are two or more inputs."

When a second layer of hidden units is added, the single layer network's bumps form disjoint regions (Principe et al. 2000). The number of neurons in the second hidden layer determines the number of bumps formed in the input space (Principe et al. 2000). Now approximations in different areas of the input space can be adjusted independently of each other. This gives an MLP with two hidden layers the ability to realize discontinuous input-output mappings.

Unfortunately, the number of parameters in a two-layer network grows very quickly. If there are h_1 and h_2 units in the first and second hidden layers respectively, and *k* (interval) inputs, the number of parameters is given by the following equation:

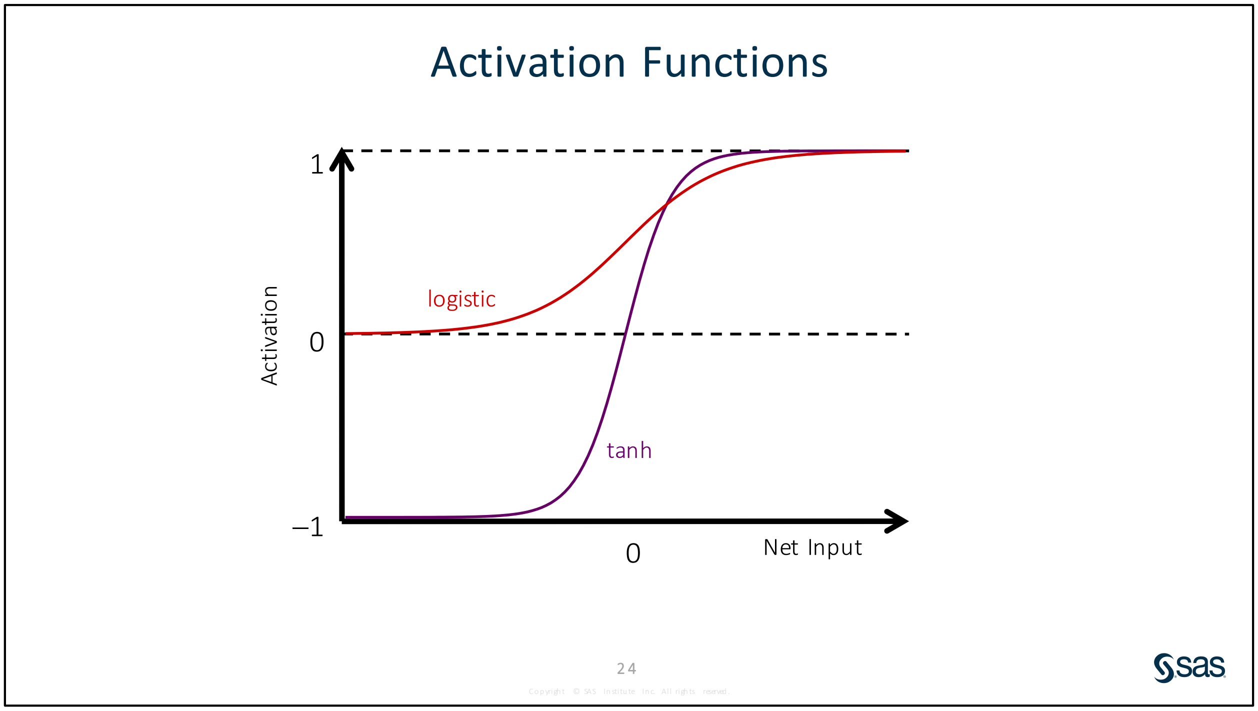
$$h_1(k+1) + h_2(h_1+1) + h_2 + 1$$

Details: More Than Two Hidden Layers

You need at most two hidden layers to approximate any function (Cybenko 1988). In fact, with gradient-based learning methods, it has been found that when more than two hidden layers are used, learning often slows to a crawl. This is known as the *vanishing gradient problem*. This problem arises because the chain rule, used to update the hidden unit weights, has the effect of multiplying *n* small numbers to compute the gradients of the front layers in an *n*-layer network. The gradient therefore decreases exponentially with *n*.

Recently, however, a major advance in neural network research known as *deep learning* (Hinton 2006) has shown that consideration of more than two hidden layers can offer substantial gains. The layers are typically trained in pairs, thereby overcoming the vanishing gradient problem. Deep learning methods transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. In order to model a target vector, a final layer of output units can be added to the top of the network, and the whole system can be fine-tuned using backpropagation.

The possible number of hidden layers in SAS Visual Data Mining and Machine Learning ranges from 0 to 10. The default is 1.



Activation Functions

24

After the weighted inputs and the bias have been combined, the neuron's net input is passed through an activation function. Many of the defining hidden unit activation functions are members of the sigmoid family. The most famous member of the sigmoid family is the logistic function:

$$\text{logistic}\left(net\right) = \frac{1}{1+e^{-net}} = \hat{p}\text{, where } net = w_0 + \sum_{i=1}^{d} w_i x_i$$

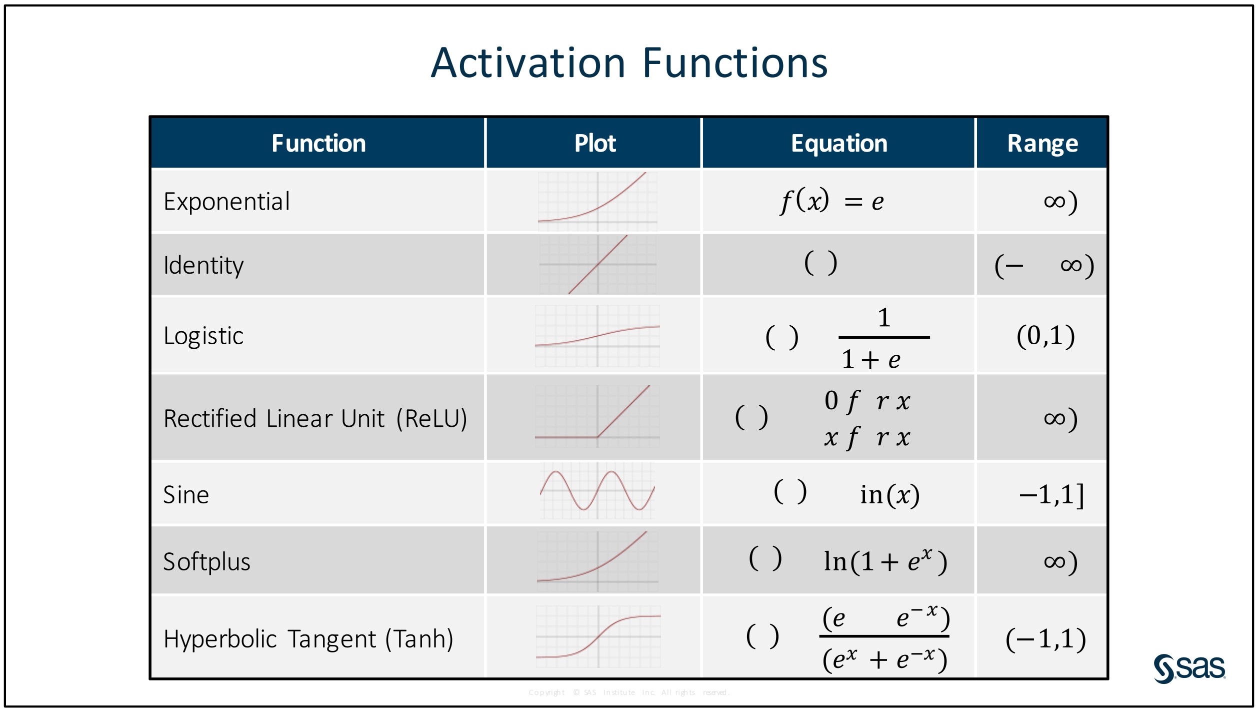
The logistic activation function constrains its output to the range 0:1, making it an ideal for generating probability ($\hat{p}$) estimates. In statistics, the logistic function is better known as the *logit-link* function:

$$\text{logit }(\hat{p}) = \ln(\frac{\hat{p}}{1-\hat{p}}) = \ln(\textit{odds})$$

Although the logistic activation function was used in early neural network research, many other sigmoidal activation functions exist. One that plays a key role in the Neural Network node is the *hyperbolic tangent* (tanh). In fact, ***it is the default hidden unit activation function***.

$$\tanh(net) = \frac{e^{net} - e^{-net}}{e^{net} + e^{-net}}$$

The hyperbolic tangent ranges from -1 to 1. This means that the inflection point is at 0 rather than at 0.5, as it is in the logistic sigmoid. This offers a small advantage during network initialization.



Activation Functions

Function	Plot	Equation	Range
Exponential		$f(x) \;\; = e$	$\infty)$
Identity		$(\;)$	$(- \quad \infty)$
Logistic		$(\;) \quad \frac{1}{1+e}$	$(0,1)$
Rectified Linear Unit (ReLU)		$(\;) \quad \begin{matrix} 0\,f\;\;r\,x \\ x\,f\;\;r\,x \end{matrix}$	$\infty)$
Sine		$(\;) \quad \text{in}(x)$	$-1,1]$
Softplus		$(\;) \quad \ln(1+e^{x})$	$\infty)$
Hyperbolic Tangent (Tanh)		$(\;) \quad \frac{(e \qquad e^{-x})}{(e^{x}+e^{-x})}$	$(-1,1)$

Interestingly, ***with respect to the hidden layer neurons, it does not seem to matter which of the sigmoid activation functions is used.*** The logistic and the hyperbolic tangent activation functions perform more or less equivalently.

Several useful non-sigmoidal activation functions are also available. For example, the *exponential* activation function generates values that range from 0 to ∞. This is particularly useful when fitting distributions that are undefined for negative input values (for example, Poisson or gamma distributions).

$$\textit{exponential}(\textit{net}) = e^{net}$$

Sigmoid and hyperbolic tangent functions have lower and upper limits. The softplus activation function returns nonnegative values. Softplus values range from zero to infinity.

$$\textit{softplus}(\textit{net}) = \ln(1+e^{net})$$

The identity activation function does not transform its argument at all. (See below.) This is useful when the desired response range is -∞ to ∞, such as when a normal target distribution is assumed.

$$\text{identity}(net) = net$$

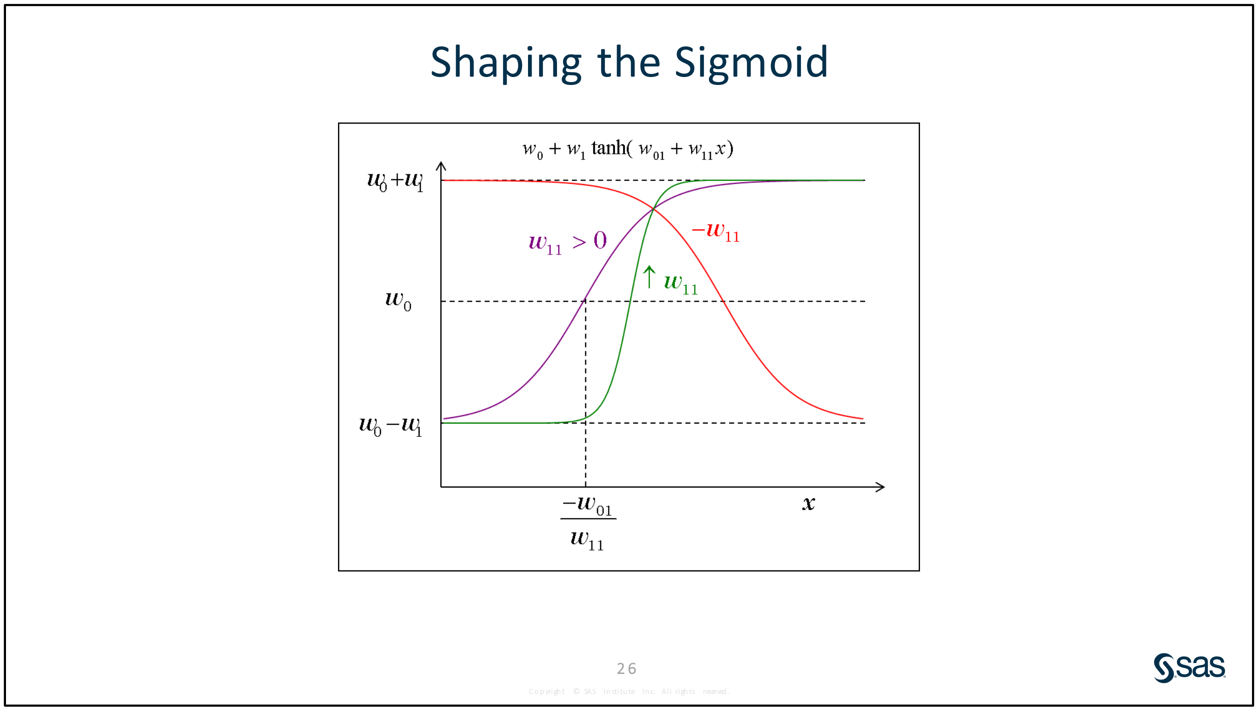
Identity is the default output activation function.

Another activation function is the rectifier. The *rectifier* has now become the *de facto* standard in neural networks. Although many variants exist, the rectifier activation function is usually defined by the following equation:

$$rectifier(net) = \max(0, net)$$

It has been argued to be more biologically plausible than the widely used ***logistic sigmoid*** and its more practical counterpart, the ***hyperbolic tangent***. A neuron using the rectifier activation function is called a *rectified linear unit* or, simply, a *rectilinear unit* (ReLU).

And finally, the sine activation function is also available. This is a well-known mathematical function. There are many ways to define the sine function. For example, the sine function can be defined using a right-angled triangle or a unit circle (LeCun et al. 1998).



Shaping the Sigmoid

26

The weights and biases give the sigmoidal surfaces their range. For example, because the maximum value returned by the hyperbolic tangent activation function is 1, the upper bound of the hyperbolic tangent's activation range is given by the output unit's bias plus its weight:

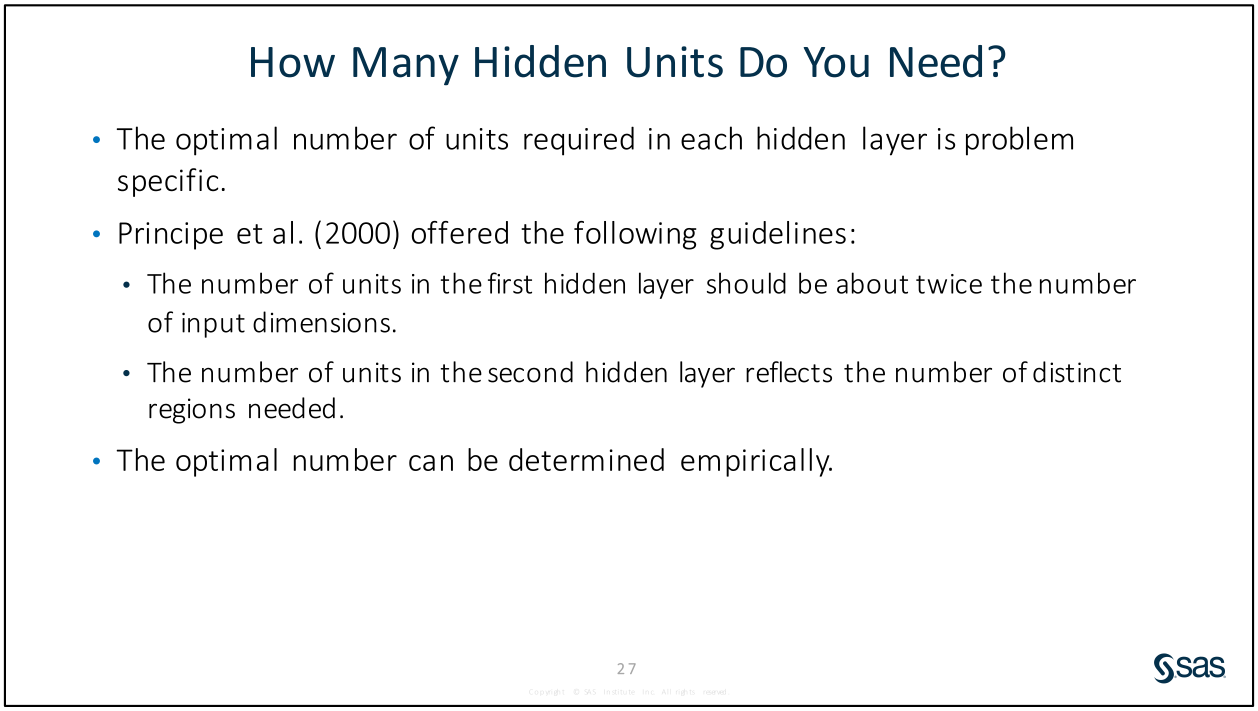
$$w_0 + w_1 \tanh(w_{01} + w_{11}x) = w_0 + w_1(1) = w_0 + w_1$$

Conversely, the minimum activation value of the hyperbolic tangent function is –1, which means that the lower bound of the hyperbolic tangent's activation range is as follows:

$$w_0 + w_1 \tanh(w_{01} + w_{11}x) = w_0 + w_1(-1) = w_0 - w_1$$

The weights and biases also give the sigmoid surfaces their flexibility. The sign of the weight associated with input *x* controls the direction of the sigmoid. Positive weight values produce the familiar s-shaped curve, and a negative weight value flips the sigmoid horizontally. The larger the absolute value of the weight, the steeper the curve. ***Steep sigmoids are often held to be responsible for overfitting***.

Note: Stopped training is one way to help keep the sigmoids from becoming too steep. This method is discussed in Section 3 of this chapter.



How Many Hidden Units Do You Need?

- The optimal number of units required in each hidden layer is problem specific.
- Principe et al. (2000) offered the following guidelines:
  - The number of units in the first hidden layer should be about twice the number of input dimensions.
  - The number of units in the second hidden layer reflects the number of distinct regions needed.
- The optimal number can be determined empirically.

27

Copyright © SAS Institute Inc. All rights reserved.

The question of the number of hidden units required is more difficult to answer than the required number of layers. If the network has too many hidden units, it will model random variation (noise) as well as the desired pattern (signal). This means that the model will fail to generalize. Conversely, having too few hidden units will fail to adequately capture the underlying signal. This means that the error value will tend to stabilize at a high value. Again, the model will fail to generalize.

Unfortunately, ***the optimal number of hidden units is problem specific***.

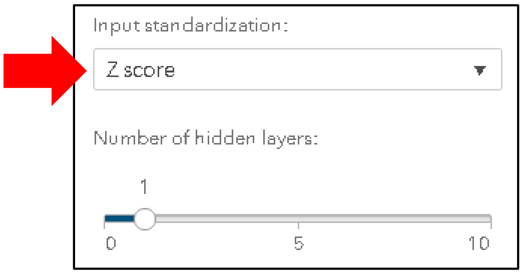
However, there are guidelines. For example, Principe et al. (2000) suggest that the number of units in the first hidden layer should be about twice the number of input dimensions. This will reflect the number of discriminant functions in the input space. If a second hidden layer is required, then the number of hidden units in the second layer should reflect the number of distinct regions needed (Principe et al. 2000).

The appropriate number of hidden units is, perhaps, best determined empirically. You start with a linear network and measure its performance on some appropriate metric, like the Schwarz-Bayesian criterion. Then increase the number of hidden units by one and observe the impact on the network's fit. Continue adding hidden units until the network's performance drops. The final network is given by the number of hidden units in the network prior to the hidden unit addition that degraded the model's performance.

Improving a Neural Network Model by Changing the Network Architecture Parameters

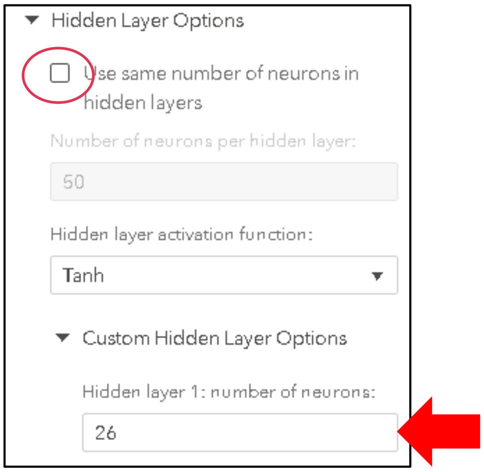
In this demonstration, you change the default settings of the Neural Network node in the Chapter 4 pipeline. You modify the network architecture parameters.

1. Recall that the average square error of the previous model, based on the default settings, was 0.0743 on the VALIDATE partition.
2. Try to improve the Neural Network performance by changing some of the default settings assigned to the network architecture.
3. Change **Input standardization** from **Midrange** to **Z score**.
4. Clear the box for **Use the same number of neurons in hidden layers**.
5. Under **Custom Hidden Layer Options**, enter **26** for **Hidden layer 1: number of neurons** (twice as many as the number of inputs).



Input standardization: Z score

Number of hidden layers: 1



Hidden Layer Options

□ Use same number of neurons in hidden layers

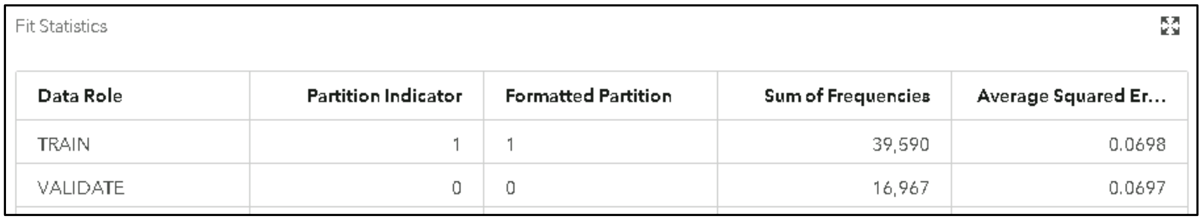
Number of neurons per hidden layer: 50

Hidden layer activation function: Tanh

Custom Hidden Layer Options

Hidden layer 1: number of neurons: 26

6. Run the Neural Network node.
7. Open the results for the Neural Network node.
8. Click the **Assessment** tab.



Fit Statistics

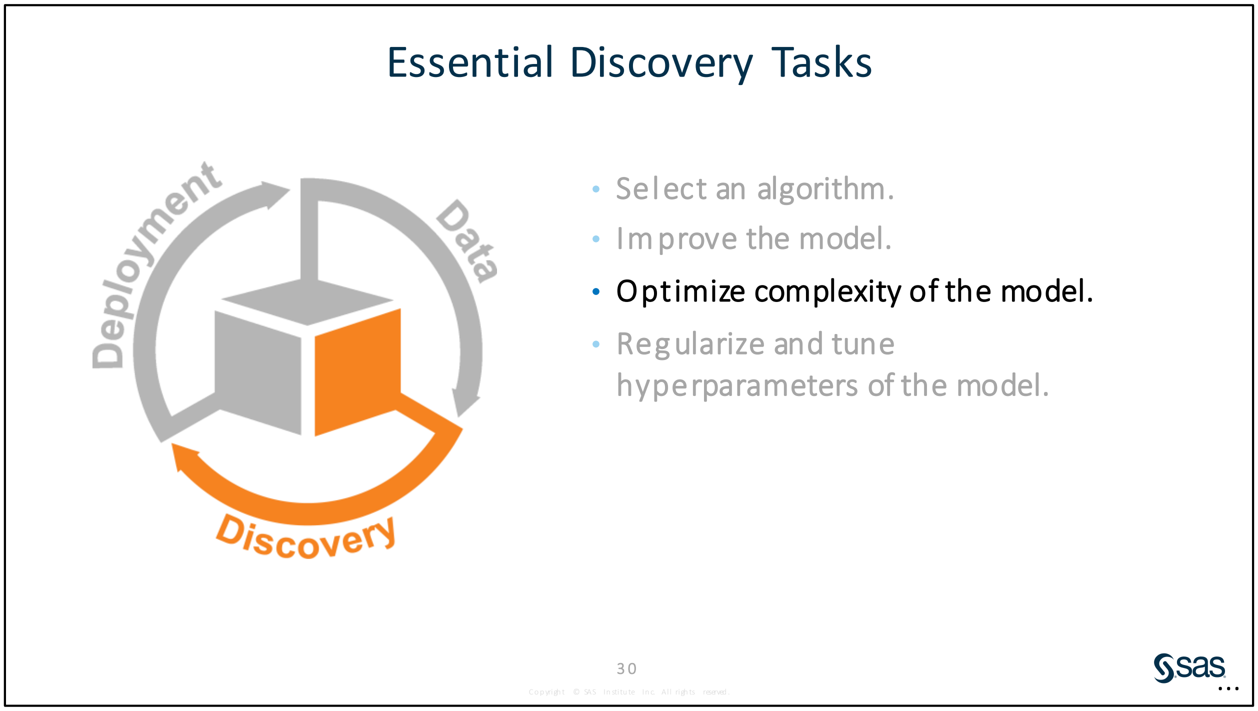
Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.0698
VALIDATE	0	0	16,967	0.0697

The average square error for the tuned Neural Network model is 0.0697 on the VALIDATE partition. This fit statistic is much better than the first model by using the default settings, by approximately 6%.

9. Close the Results window.

End of Demonstration

4.3 Learning

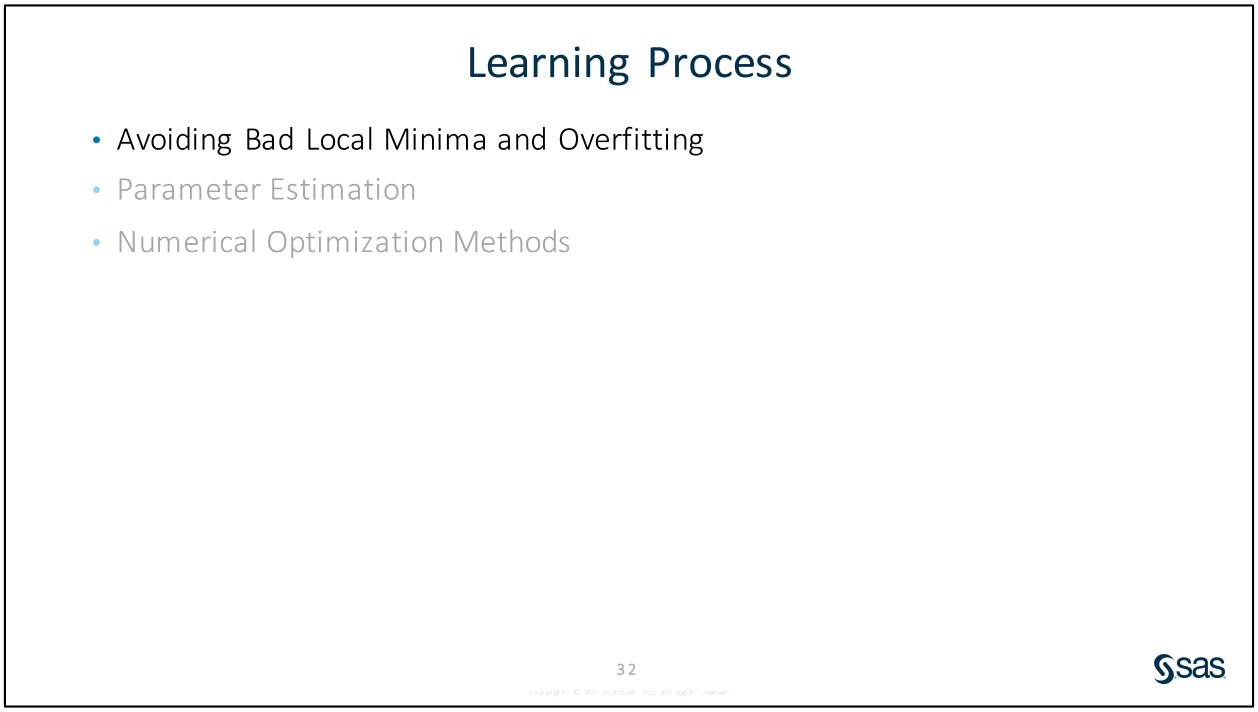

Deployment
Data
Discovery

Essential Discovery Tasks

- Select an algorithm.
- Improve the model.
- Optimize complexity of the model.
- Regularize and tune hyperparameters of the model.

30

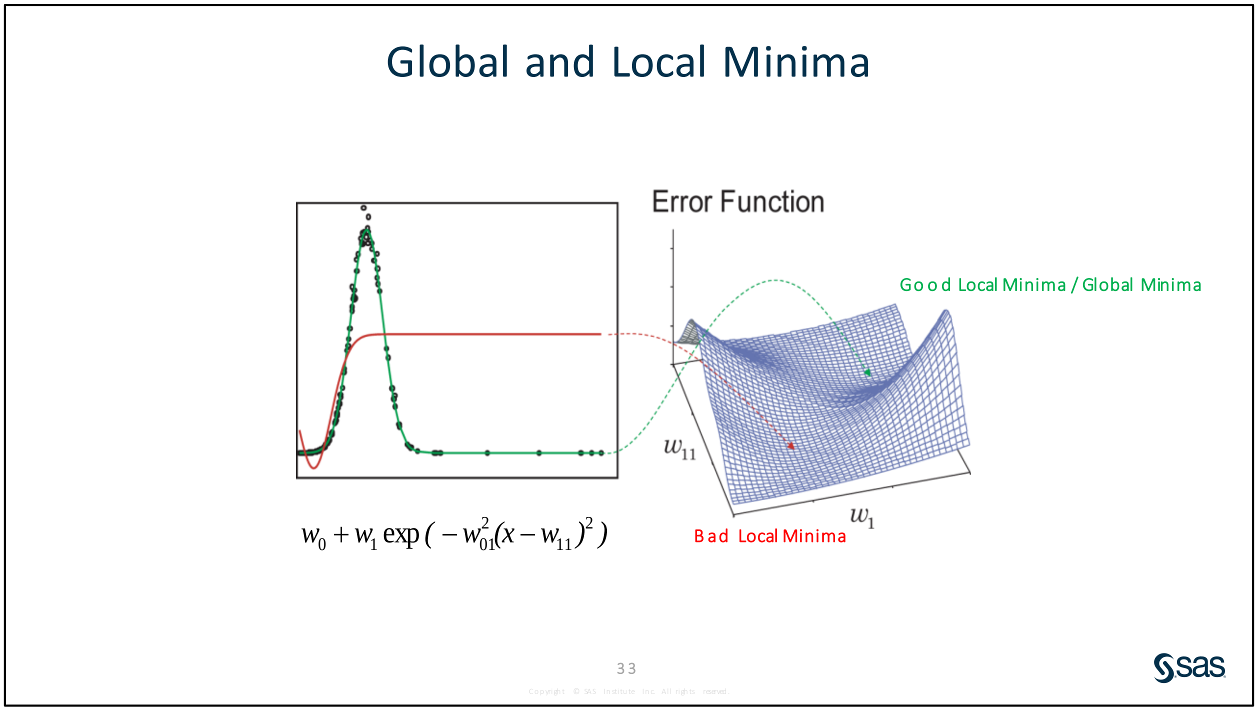
The way in which neural network models optimize complexity is very different from other algorithms. For example, there is not a clear "sequence of models that increase in complexity" in the same way there is for decision trees and regression models using a stepwise method. For neural networks, complexity is optimized through the learning process and the numerical methods used to search for updates in weight estimates. Stopped training is also a method to optimize complexity, but it is mostly used to avoid overfitting. These methods to optimize complexity are discussed in this section.



Learning Process

- Avoiding Bad Local Minima and Overfitting
- Parameter Estimation
- Numerical Optimization Methods

Two steps are very important in a neural network. The first one is to find a good set of parameters that minimizes the error (avoid bad local minima). The second one is to ensure that this set of parameters performs well (minimize the error) in different (new) data sets (avoid overfitting).

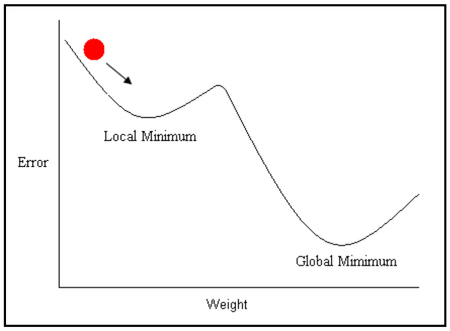


Global and Local Minima

$$w_0 + w_1 \exp(-w_{01}^2(x - w_{11})^2)$$

A *global minimum* is a set of weights generating the smallest amount of error. A simple strategy to ensure that a global minimum has been attained is a brute-force search of the parameter space. Unfortunately, the curse of dimensionality quickly makes this method infeasible.

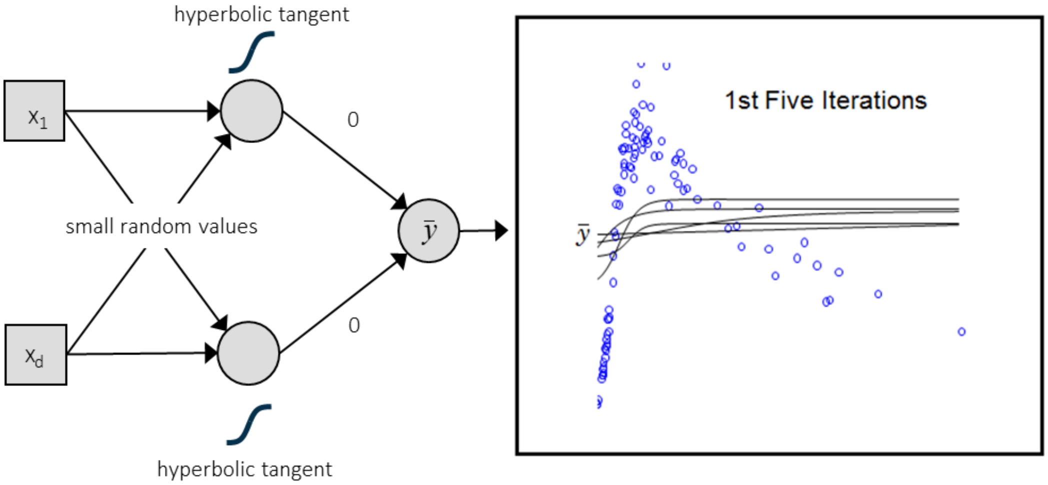
Many optimization algorithms are not guaranteed to converge to a global error minimum. Rather, many search optimization algorithms are ***heuristic*** methods that key on ***local*** features of the error surface when making their decisions. This makes them vulnerable to *local minima*—that is, areas of the error surface generating non-optimal error values. (See the diagram below.)



From the optimization algorithm's perspective, when a local minimum is reached, any movement away from the bottom leads to an increase in error. Because this is unacceptable, the search stops.

Unlike the parabolic error surface of a generalized linear model fit using least squares (which has no local minima), the error surface of a nonlinear model is plagued with local minima. Fortunately, many of these local minima have nearly the global error value. It is only the worst of them that must be avoided.

Details: Initialization Procedure

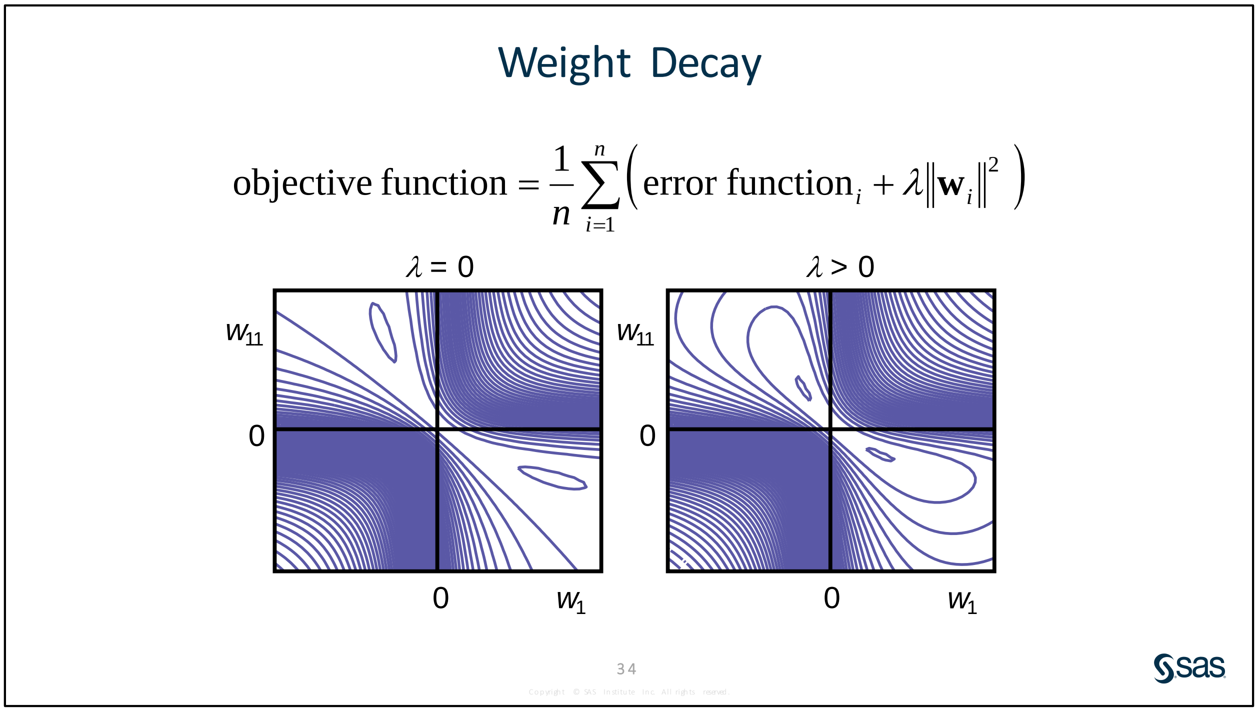


One way to avoid the worst local minima is to start with good weight values. In the Neural Network task, this is accomplished by means of the following five-step initialization process:

1. Standardize the inputs to have a midrange of 0, a minimum value of -1, and a maximum value of 1 (midrange).
2. Set the input-to-hidden weights to a small random number.
3. Use the hyperbolic tangent activation function so that the inflection point is at zero. The Elliott or arctangent functions are acceptable alternatives because they also have a zero inflection point.
4. Set the hidden-to-output connection weights to zero.

5. Because the hidden-to-output weights are set to zero in step 4, the output activation is given solely by the output bias on the first iteration. Therefore, the output bias is initialized to the mean of the target.

These initializations help prevent the optimization algorithms from stepping into treacherous regions of the parameter space, that is, regions of the parameter space with many bad local minima.



Weight Decay

$$\text{objective function} = \frac{1}{n}\sum_{i=1}^{n}\left(\text{error function}_i + \lambda\left\|\mathbf{w}_i\right\|^2\right)$$

Bartlett (1997) demonstrated that generalization depends more on the magnitude of the weights than on the number of weights. Very large magnitude weights tend to generate an irregular fit to the data, as the model adapts to noise (random variation). In other words, large weights are responsible for overfitting.

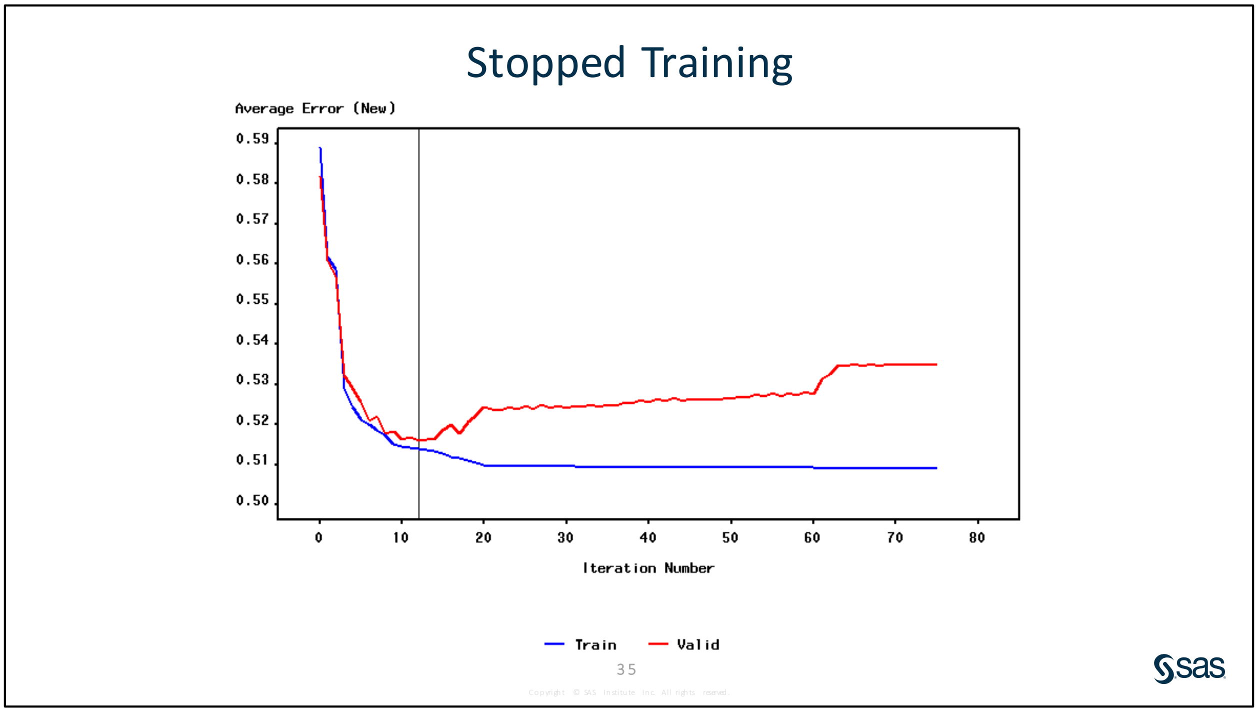
Weight decay is one way to keep the weights from growing too large. It penalizes large weights by adding a term that is proportional to the sum of squared weights (Bishop 1995). That is,

$$\text{objective function} = \frac{1}{n}\sum_{i=1}^{n}\left(\text{error function}_i + \lambda\left\|\mathbf{w}_i\right\|^2\right)$$

The decay parameter λ, which can range from zero to one, controls the relative importance of the penalty term. Specifying too large of a penalty term risks the model under-fitting the data. However, as Ripley (1996) points out, the advantages of weight decay far outweigh its risks [emphasis added]:

> Weight decay helps the optimization in several ways. When weight decay terms are included, it is normal to find fewer local minima, and as the objective function is more nearly quadratic, the quasi-Newton and conjugate gradient methods exhibit super-linear convergence in many fewer iterations. ***There seems no reason to ever exclude a regularizer such as weight decay***. (pp. 159-160)

In any event, the influence of the penalty term is usually kept extremely small. In many published studies, the magnitude of the decay parameter (λ) is on the order of 0.000001.

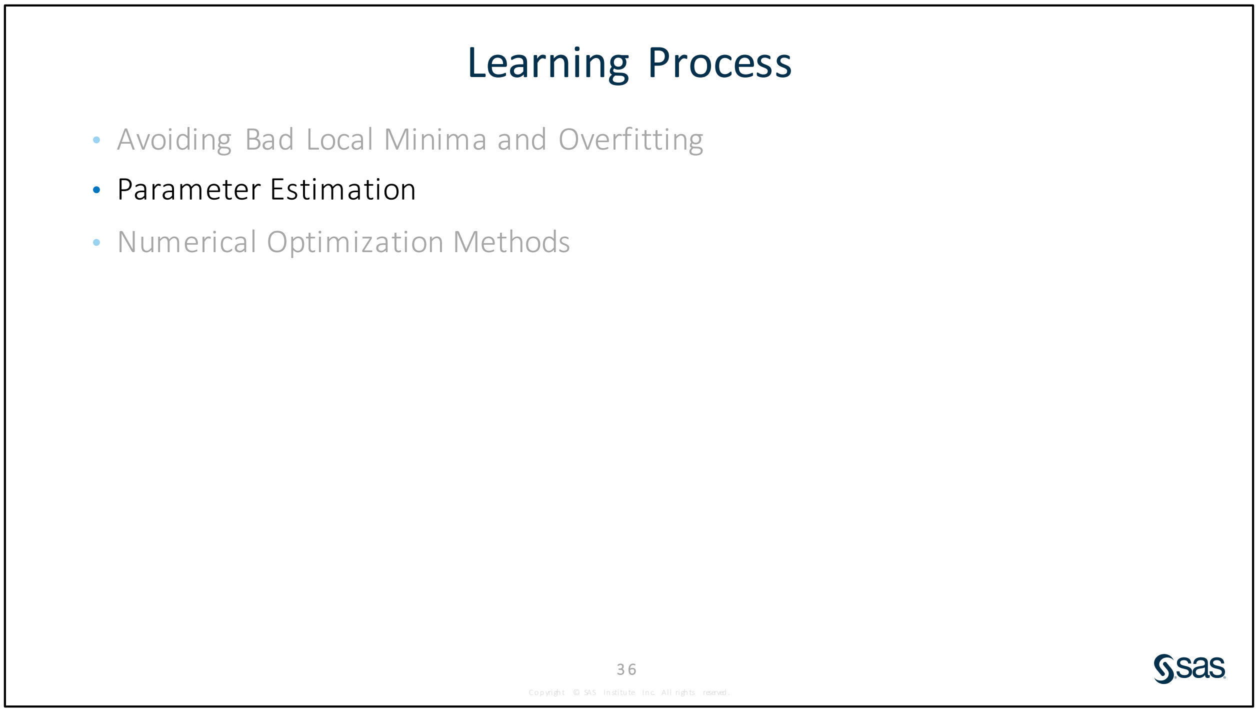


Stopped Training

35

Stopped training is another way to keep the weights from growing too large. It is closely related to ridge regression (Sarle 1995). In SAS Viya, training starts with a pseudo randomly generated set of initial weights. The NNET procedure then computes the objective function for the training partition, and the optimization algorithm adjusts the weights. This process is repeated until any one of the following conditions is met:

- The objective function that is computed using the training partition stops improving.
- The objective function that is computed using the validation partition stops improving.
- The process has been repeated the number of times specified in the MAXITER= and MAXTIME= options in the OPTIMIZATION statement.

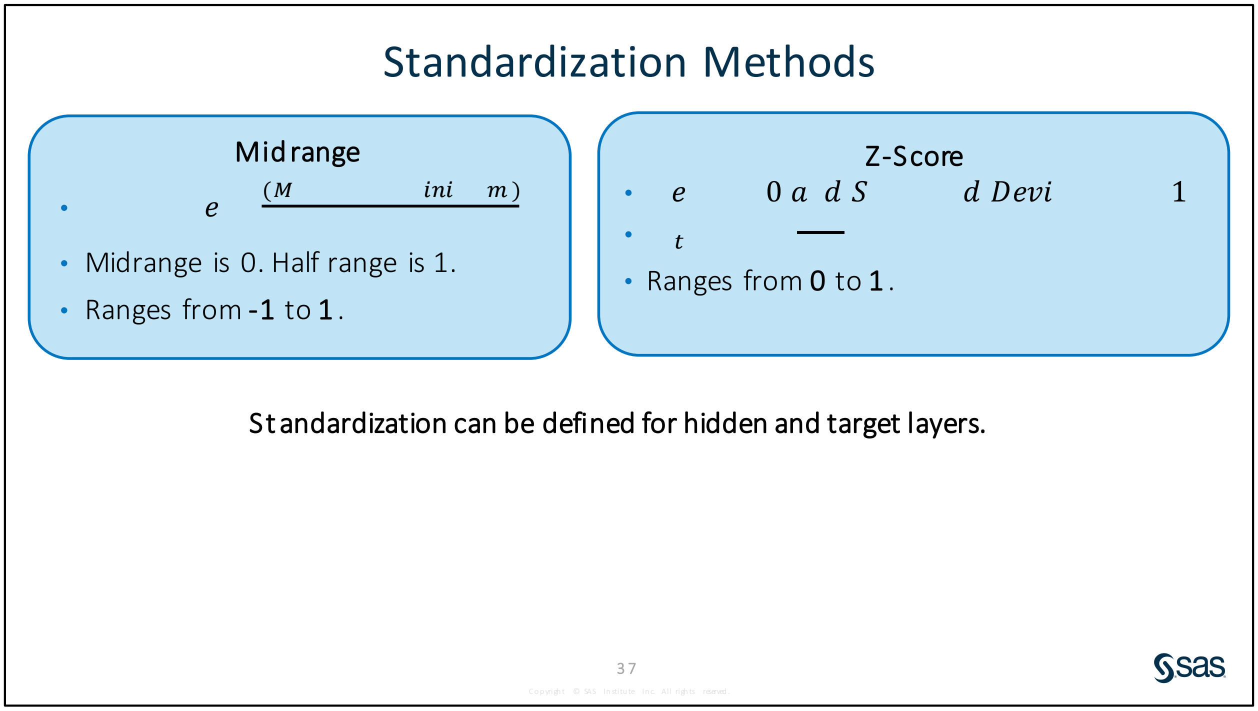


Learning Process

- Avoiding Bad Local Minima and Overfitting
- Parameter Estimation
- Numerical Optimization Methods

36

The error function defines the surface of the parameter space. In this way, the neural network learns or searches for the best set of weights to minimize the error, depends on the type of the surface. Then, the numerical method to estimate the weights is based on the error function.



Standardization Methods

Midrange

- $e \quad \frac{(M \quad ini \quad m)}{}$
- Midrange is 0. Half range is 1.
- Ranges from **-1** to **1**.

Z-Score

- $e \quad 0\, a \;\; d\, S \quad d\; Devi \quad 1$
- t
- Ranges from **0** to **1**.

Standardization can be defined for hidden and target layers.

37

Standardizing means to rescale your data to have a mean of zero and a standard deviation of 1. The value of a standardized variable is sometimes called a z-score or a standard score. Here is the equation to standardize your data:

$$x_{std} = z = \frac{x - \mu}{\sigma}$$

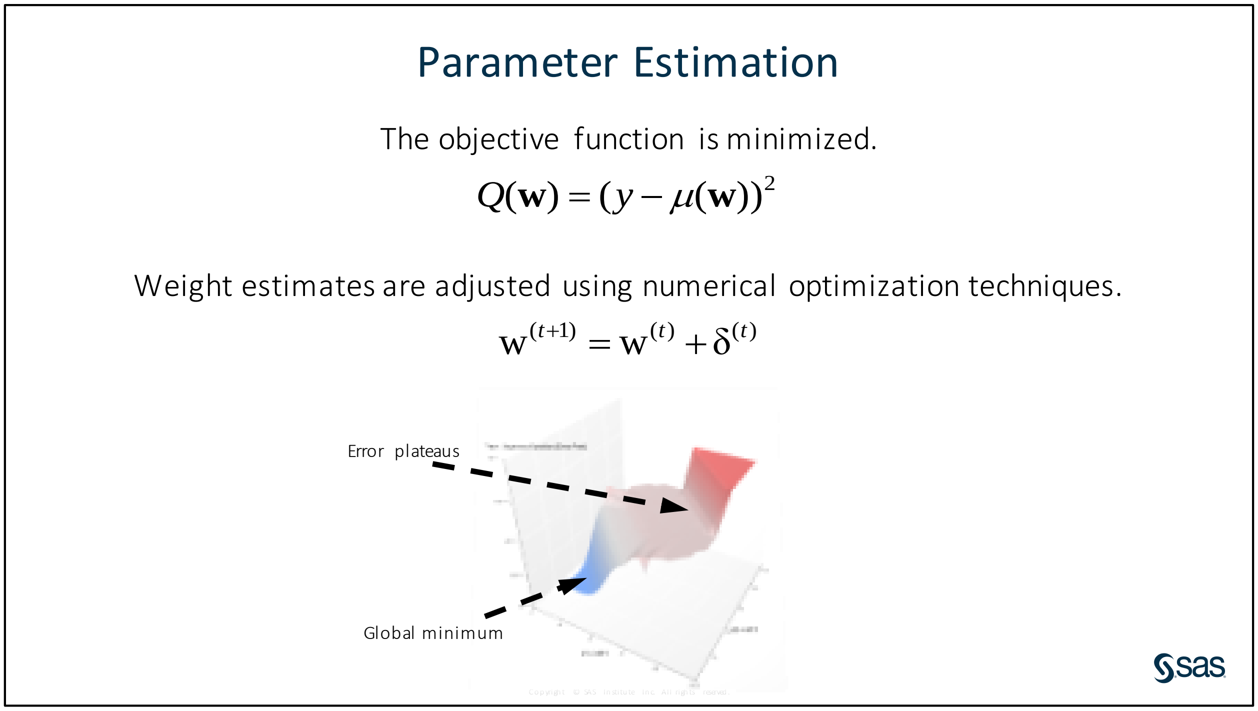
where x is the original value, μ is the variable's mean, and σ is the variable's standard deviation. The absolute value of z represents the distance between the raw score and the population mean in units of the standard deviation. z is negative when the original value is below the mean, positive when above.

Normalizing is another rescaling method with many meanings in statistics and statistical applications. Most commonly, normalizing rescales numeric data between zero and 1 using the following equation:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

where x_{min} is the variable's minimum value, and x_{max} is the variable's maximum value.

Model stability and parameter estimate precision are influenced during multivariate analysis when multi-scaled variables are used. For example, in boundary detection, a variable that ranges between 0 and 100 will outweigh a variable that ranges between 0 and 1. Using variables without standardization can give variables with larger ranges greater importance in the analysis. Transforming the data to comparable scales can prevent this problem.



Parameter Estimation

The objective function is minimized.

$$Q(\mathbf{w}) = (y - \mu(\mathbf{w}))^2$$

Weight estimates are adjusted using numerical optimization techniques.

$$\mathrm{w}^{(t+1)} = \mathrm{w}^{(t)} + \delta^{(t)}$$

A *global minimum* is a set of weights generating the smallest amount of error. A simple strategy to ensure that a global minimum has been attained is a brute-force search of the parameter space. Unfortunately, the curse of dimensionality quickly makes this method infeasible.

Search optimization algorithms are *heuristic* methods that key on ***local*** features of the error surface when making their decisions. This makes them vulnerable to *error plateaus* (that is, areas of the error surface generating non-optimal error values). An error plateau is an area of the error space in which very little improvement is attained, given the current dot product of the inputs and weights. Previously, error plateaus were viewed as *local minima.* It is possible for a model to get stuck at a local minima, but this will occur only if the process has arrived at a saddle point for each model degree of freedom. Therefore, local minima are highly unlikely in higher dimensional spaces.

From the optimization algorithm's perspective, when an error plateaus is reached, any further movements yield very little improvement in the error. And at this point, the search stops.

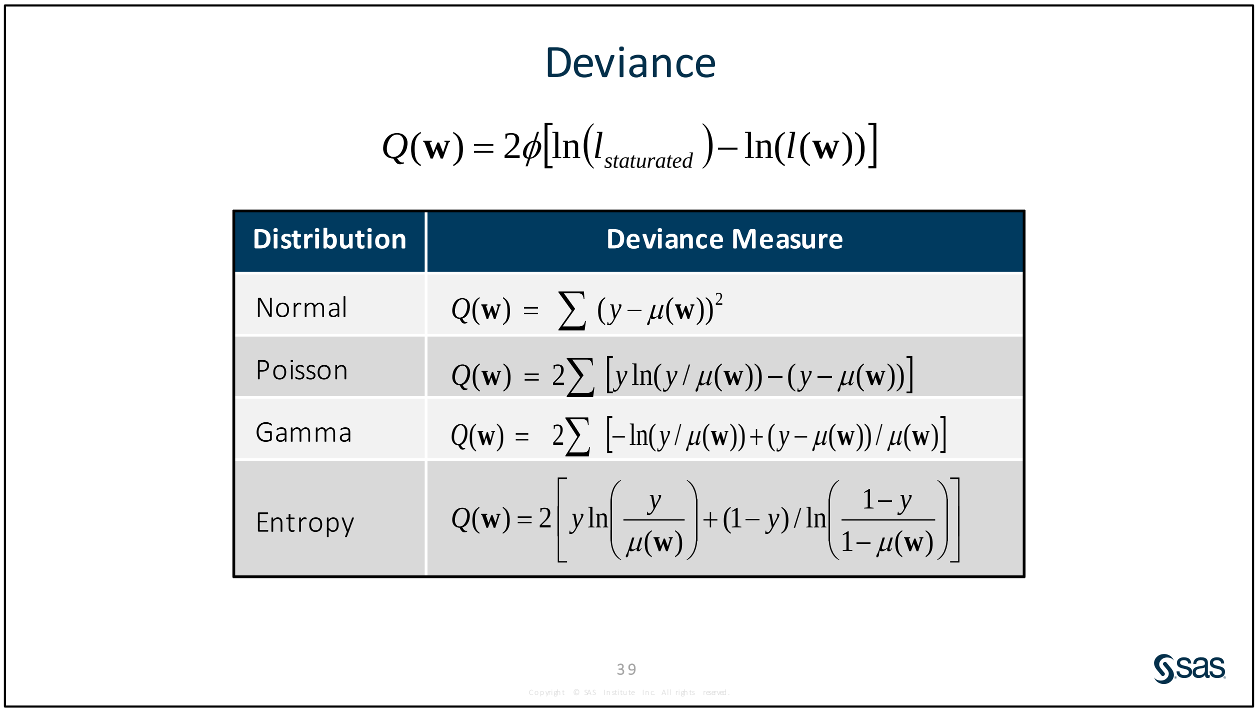
Unlike the parabolic error surface of a generalized linear model fit using least squares (which has no local minima), the error surface of a nonlinear model is plagued with error plateaus. Fortunately, many of these error plateaus have nearly the global error value. It is only the worst of them that must be avoided.

To efficiently search this landscape for an error minimum, optimization must be used. The optimization methods use local features of the error surface to guide their descent. Specifically, the weights associated with a given error minimum are located using the following procedure:

1. Initialize the weight vector to small random values, $\mathbf{w}^{(0)}$.
2. Use an optimization method to determine the update vector, $\delta^{(t)}$.
3. Add the update vector to the weight values from the previous iteration to generate new estimates:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \boldsymbol{\delta}^{(t)}$$

4. If none of the specified convergence criteria have been achieved, then go to step 2.



Deviance

$$Q(\mathbf{w}) = 2\phi\left[\ln\left(l_{staturated}\right) - \ln(l(\mathbf{w}))\right]$$

Distribution	Deviance Measure
Normal	$Q(\mathbf{w}) = \sum (y-\mu(\mathbf{w}))^2$
Poisson	$Q(\mathbf{w}) = 2\sum \left[y\ln(y/\mu(\mathbf{w})) - (y-\mu(\mathbf{w}))\right]$
Gamma	$Q(\mathbf{w}) = 2\sum \left[-\ln(y/\mu(\mathbf{w})) + (y-\mu(\mathbf{w}))/\mu(\mathbf{w})\right]$
Entropy	$Q(\mathbf{w}) = 2\left[y\ln\left(\frac{y}{\mu(\mathbf{w})}\right) + (1-y)/\ln\left(\frac{1-y}{1-\mu(\mathbf{w})}\right)\right]$

39

If the probability distribution is a member of the exponential family, minimizing deviance is equivalent to maximizing likelihood. But deviance offers the advantage that it does not require a probability density function, which makes the calculation of deviance more efficient. Deviance also offers other numerical advantages. For example, the deviance measures are automatically scaled.

The default method for fitting an interval target is a *normal* distribution. The deviance measure used to fit a normal distribution is the familiar ordinary least squares equation (below):

$$Q(\mathbf{w}) = \sum (y - \mu(\mathbf{w}))^2$$

A *Poisson* distribution is usually thought of as the appropriate distribution for count data. Because the variance is proportional to the mean, the deviance function for a Poisson distribution is the following:

$$Q(\mathbf{w}) = 2\sum [y\ln(y/\mu(\mathbf{w})) - (y - \mu(\mathbf{w}))]$$

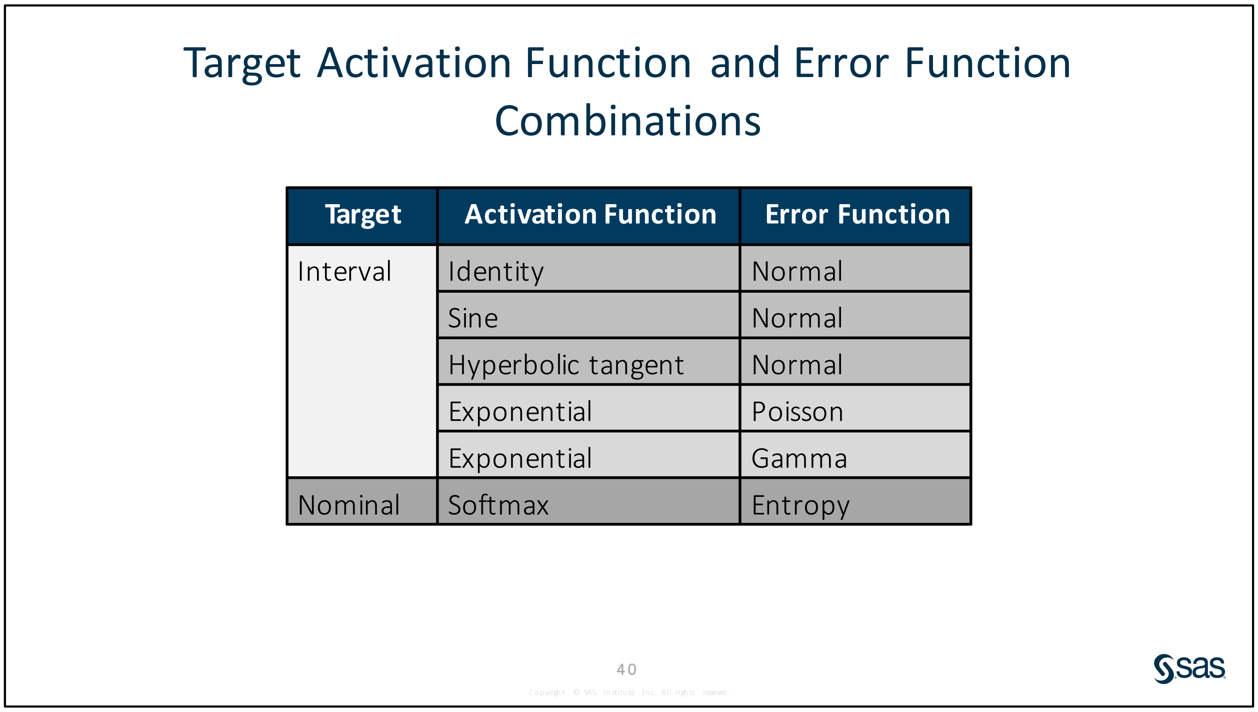
In a *gamma* distribution, the variance is proportional to the square of the mean. It is often used when the target represents an amount. The gamma deviance function is given by the following:

$$Q(\mathbf{w}) = 2\sum\left[-\ln(y/\mu(\mathbf{w})) + (y - \mu(\mathbf{w}))/\mu(\mathbf{w})\right]$$

Entropy

Cross or relative entropy is for independent interval targets with values between zero and 1 (inclusive). ***Identical to the Bernoulli distribution if the target is binary***, it offers some advantages over the Bernoulli distribution when the data are proportions. The entropy deviance estimate is given by the following:

$$Q(\mathbf{w}) = 2\left[y\ln\left(\frac{y}{\mu(\mathbf{w})}\right) + (1-y)/\ln\left(\frac{1-y}{1-\mu(\mathbf{w})}\right)\right]$$

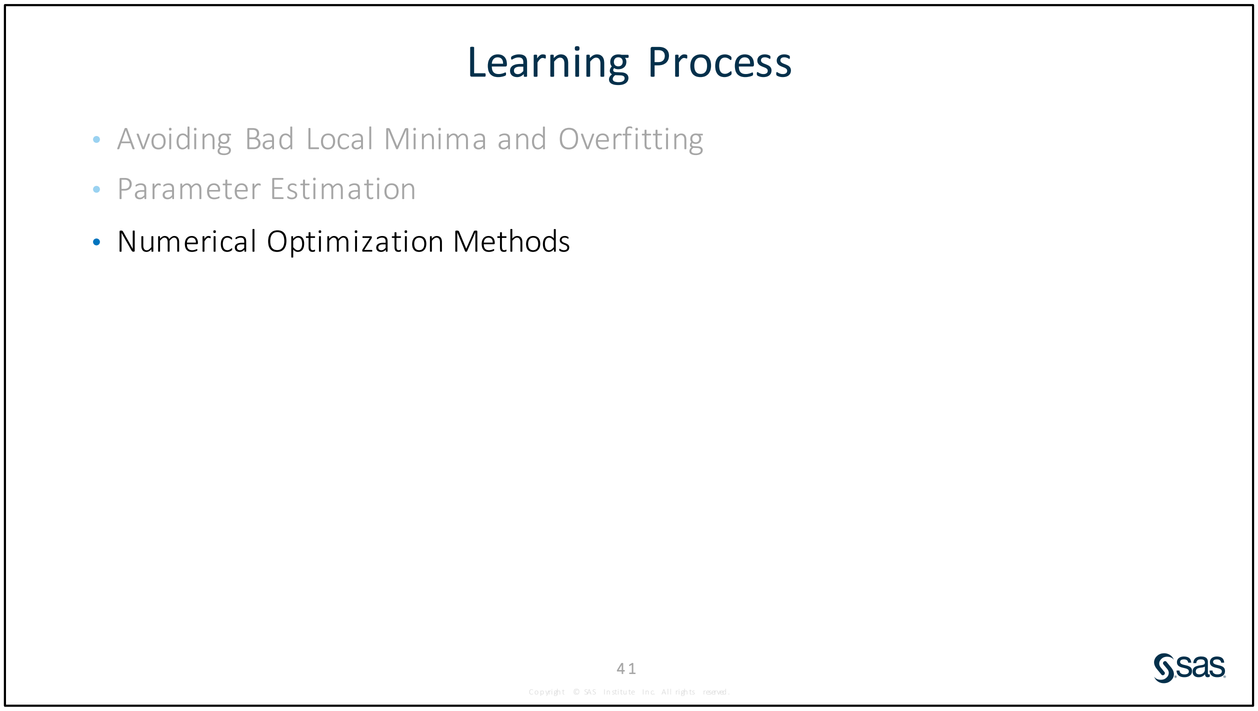


Target Activation Function and Error Function Combinations

Target	Activation Function	Error Function
Interval	Identity	Normal
	Sine	Normal
	Hyperbolic tangent	Normal
	Exponential	Poisson
	Exponential	Gamma
Nominal	Softmax	Entropy

The slide above summarizes the appropriate activation and error function combinations.

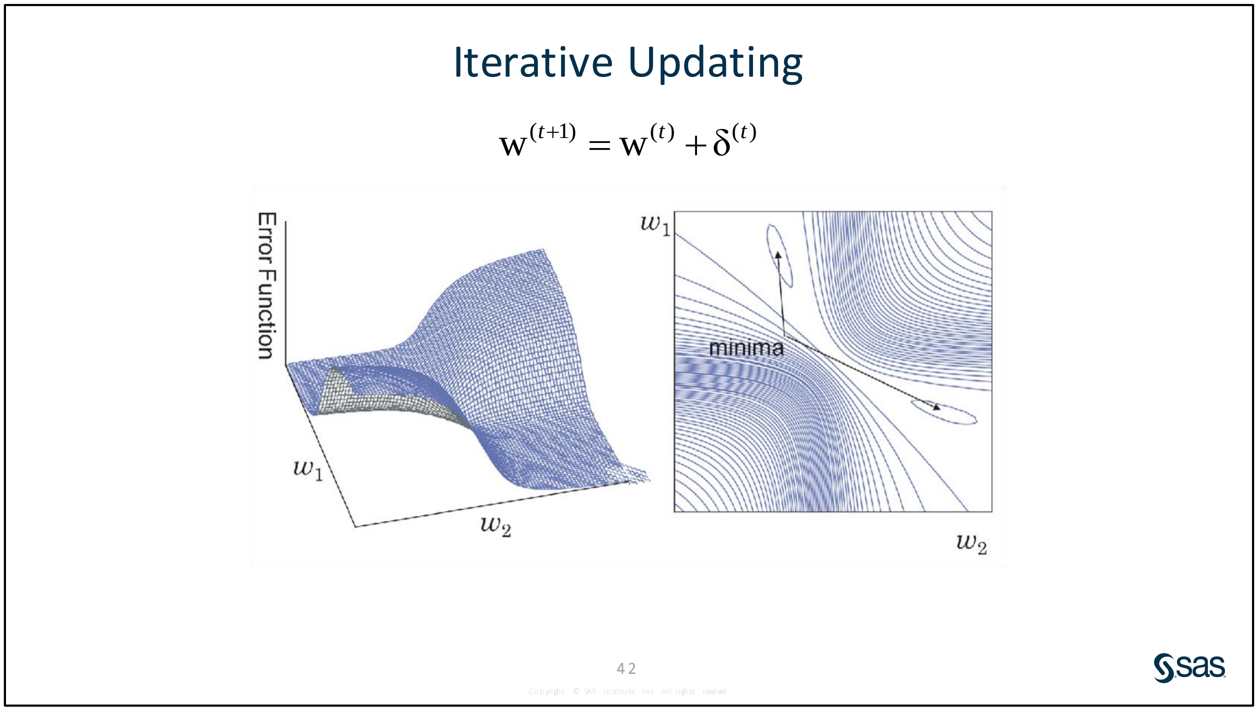
If the target is interval, there are four possible activation functions to be used: Identity, Sine, Hyperbolic Tangent, and Exponential. If the target distribution is normal, the error function should be Normal. If the target distribution is exponential, the error function can be Poisson or Gamma. If the target is nominal, the activation functional should be Softmax and the error function should be Entropy.



Learning Process

- Avoiding Bad Local Minima and Overfitting
- Parameter Estimation
- Numerical Optimization Methods

There are two optimization methods available in the Neural Network node of Model Studio: limited memory BFGS and stochastic gradient descent. They are both discussed briefly over the next few slides.



Iterative Updating

$$w^{(t+1)} = w^{(t)} + \delta^{(t)}$$

The error function defines a surface in the parameter space. If it is a linear model fit by least squares, the error surface is a parabola. However, in a nonlinear model, this error surface is often a complex landscape consisting of numerous deep valleys and steep cliffs.

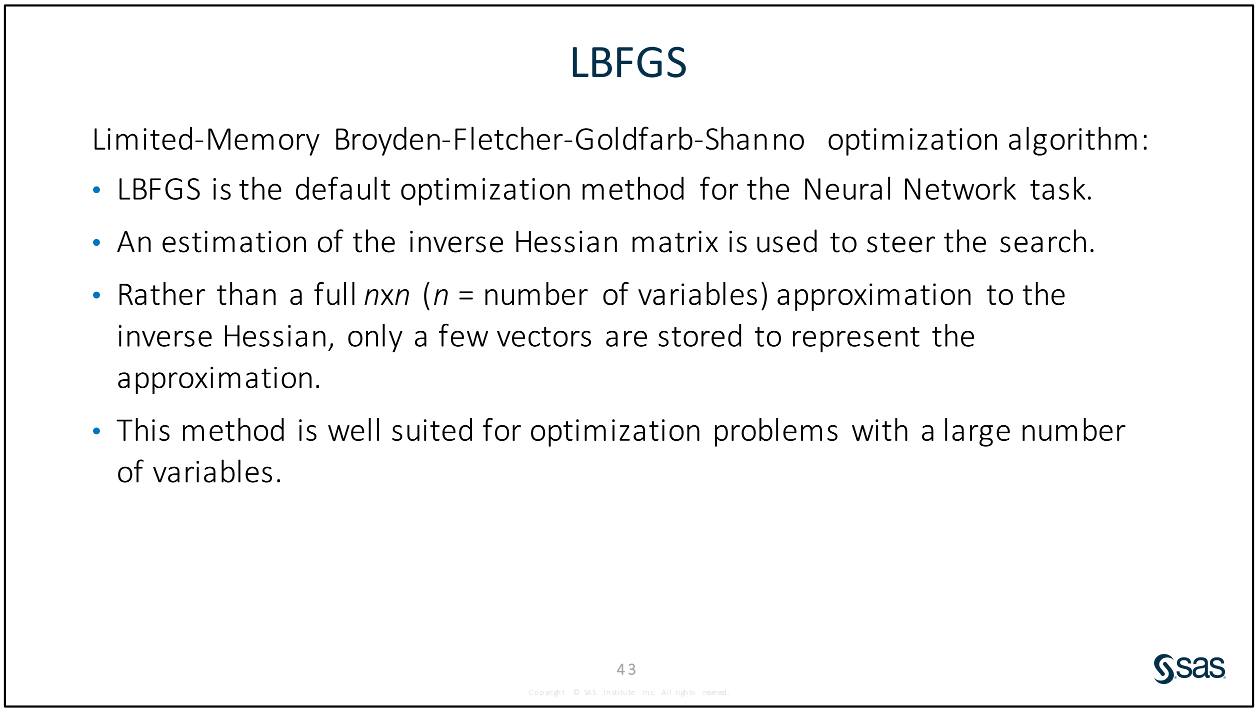
To efficiently search this landscape for an error minimum, optimization must be used. The optimization methods use local features of the error surface to guide their descent. Specifically, the weights associated with a given error minimum are located using the following procedure:

1. Initialize the weight vector to small random values, $\mathbf{w}^{(0)}$.
2. Use an optimization method to determine the update vector, $\delta^{(t)}$.
3. Add the update vector to the weight values from the previous iteration to generate new estimates:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \boldsymbol{\delta}^{(t)}$$

4. If none of the specified convergence criteria have been achieved, then go to step 2.

In this topic, you examine few ways of determining the update vector $\delta^{(t)}$ used in step 2 above.



LBFGS

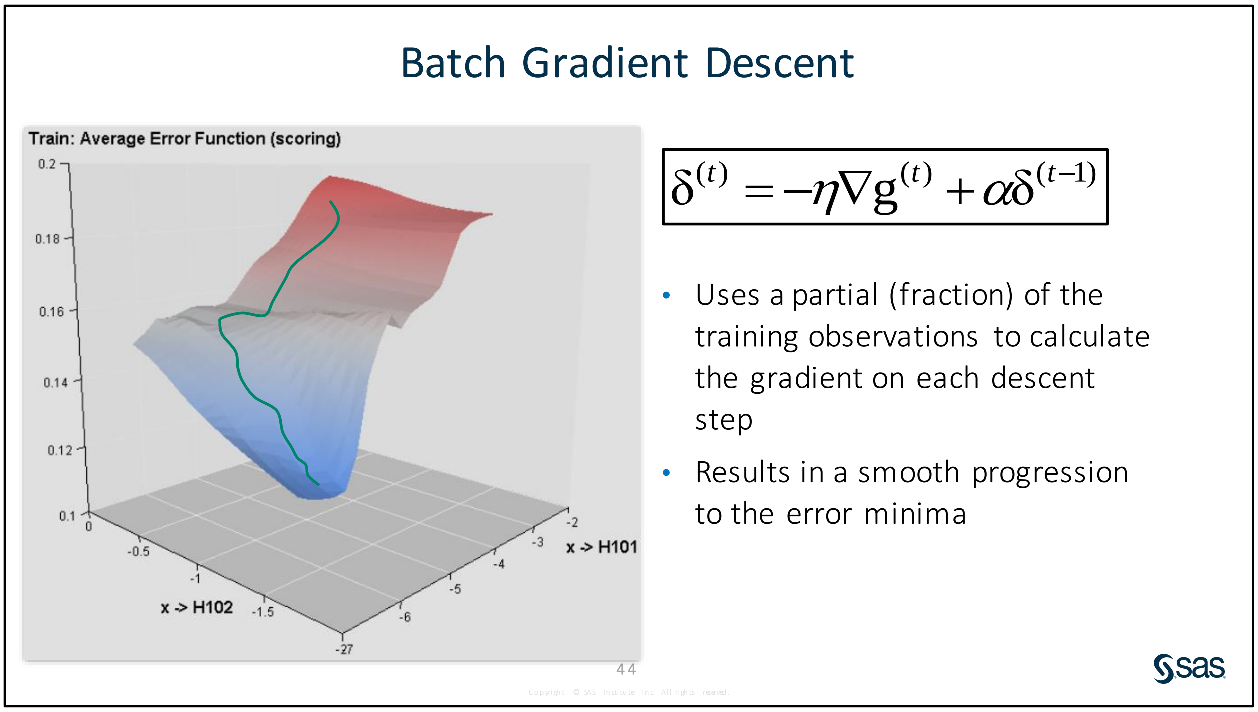
Limited-Memory Broyden-Fletcher-Goldfarb-Shanno optimization algorithm:

- LBFGS is the default optimization method for the Neural Network task.
- An estimation of the inverse Hessian matrix is used to steer the search.
- Rather than a full *n*x*n* (*n* = number of variables) approximation to the inverse Hessian, only a few vectors are stored to represent the approximation.
- This method is well suited for optimization problems with a large number of variables.

43

The default optimization method in the neural network task in SAS Visual Data Mining and Machine Learning is a variant of the BFGS method known as *limited memory BFGS*. Like the original BFGS, the limited memory BFGS (L-BFGS) uses an estimation of the inverse Hessian to steer the search. But, whereas BFGS stores an *n* by *n* approximation to the Hessian (where *n* is the number of variables), the L-BFGS variant stores only a few vectors that represent the approximation implicitly. Thus, L-BFGS is well suited for optimization problems with a large number of variables (Byrd et al. 1995).

The LBFGS is an optimization algorithm in the family of quasi-newton methods that approximates the BFGS algorithm using only some specific gradients to represent the approximation implicitly. It uses less computer memory due to its linear memory requirement. The algorithm starts with initial estimates of the optimal value of weights and progresses continuously to improve the estimates of the weights. The derivatives of the function of the estimates are used to drive the algorithm to find the direction of the steepest descent. The derivatives are also used to find the estimate of the Hessian matrix (second derivative).



Batch Gradient Descent

$$\boxed{\delta^{(t)} = -\eta \nabla g^{(t)} + \alpha \delta^{(t-1)}}$$

- Uses a partial (fraction) of the training observations to calculate the gradient on each descent step
- Results in a smooth progression to the error minima

Re-invented several times, the back propagation (*backprop*) algorithm initially just used *gradient descent* to determine an appropriate set of weights. The gradient, $\nabla \mathbf{g}^{(t)}$, is the vector of partial derivatives of the error function with respect to the weights; it points in the steepest direction uphill. By negating the step size (that is, *learning rate*) parameter, η, a step is made in the direction that is locally steepest downhill.

$$\boldsymbol{\delta}^{(t)} = -\eta \nabla \mathbf{g}^{(t)}$$

The weights associated with a given error minimum are located using the following procedure:

1. Initialize the weight vector to small random values, $\mathbf{w}^{(0)}$.
2. Use an optimization method to determine the update vector, $\delta^{(t)}$.
3. Add the update vector to the weight values from the previous iteration to generate new estimates:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \boldsymbol{\delta}^{(t)}$$

4. If none of the specified convergence criteria has been achieved, then go to step 2.

Unfortunately as gradient descent approaches the desired weights, it exhibits numerous back-and-forth movements known as *hemstitching*. To control the training iterations wasted in this hemstitching, later versions of back propagation included a momentum term, yielding the modern update rule:

$$\boldsymbol{\delta}^{(t)} = -\eta \nabla \mathbf{g}^{(t)} + \alpha \boldsymbol{\delta}^{(t-1)}$$

The momentum term retains the last update vector, $\delta^{(t\text{-}1)}$, using this information to “dampen” potentially oscillating search paths. The cost is an extra learning rate parameter (0 ≤ α ≤ 1) that must be set.

Note: The default value of α is 0. This means that, by default, backprop performs gradient descent.

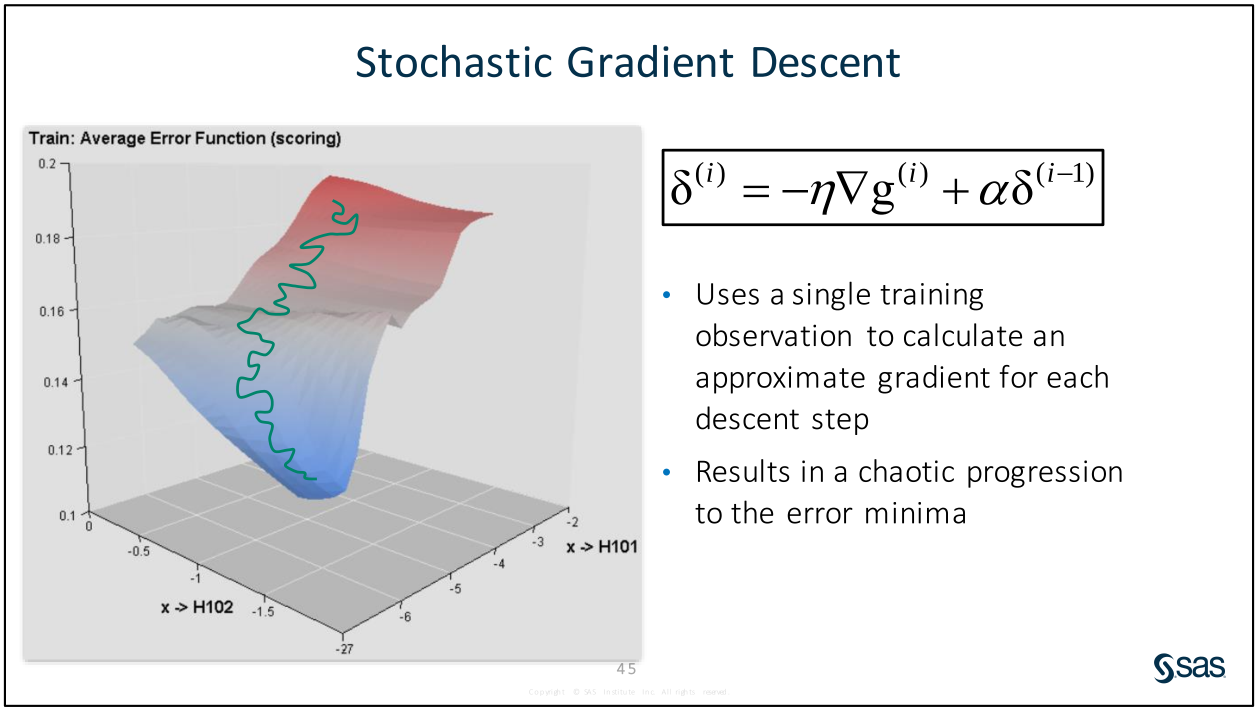
In the (default) batch variant of the gradient descent algorithm, generation of the weight update vector is determined by using all of the examples in the training set. That is, the exact gradient is calculated, ensuring a relatively smooth progression to the error minima.

Note: For a linear neuron, with squared error, the error surface is a quadratic bowl. The vertical cross-sections are parabolas, and the horizontal cross-sections are ellipses. For multilayer networks, the error surface is much more complicated. But provided that the weights are not too big, locally the error surface can be well approximated by a piece of a quadratic bowl (Hinton 2013).

However, when the training data set is large, computing the exact gradient is computationally expensive. The entire training data set must be assessed on each step down the gradient. Moreover, if the data is redundant, the error gradient on the second half of the data will be almost identical to the gradient on the first half. In this event it would be a waste of time to compute the gradient on the whole data set. You would be better off computing the gradient on a subset of the weights, updating the weights, and then repeat on a new subset. In this case, each weight update is based on an approximation to the true gradient. But as long as it points in approximately the same direction as the exact gradient, the approximate gradient is a useful alternative to computing the exact gradient (Hinton 2007).

A compromise between batch gradient descent and single-case stochastic gradient descent is to divide the training data into small batches, compute the gradient using a single batch, make an update, and then move to the next batch of observations. This is known as *mini-batch gradient descent*.

Like single-case stochastic gradient descent, mini-batch gradient descent is typically faster than batch gradient descent. And because the weights are updated less often than when single-observation stochastic gradient descent is used, mini-batch gradient descent typically uses less computation updating the weights, that is, each mini-batch computes the gradients for a number of cases in parallel. However, it is important that the mini-batches contain approximately balanced classes (Hinton 2013).



Stochastic Gradient Descent

$$\boxed{\delta^{(i)} = -\eta \nabla g^{(i)} + \alpha \delta^{(i-1)}}$$

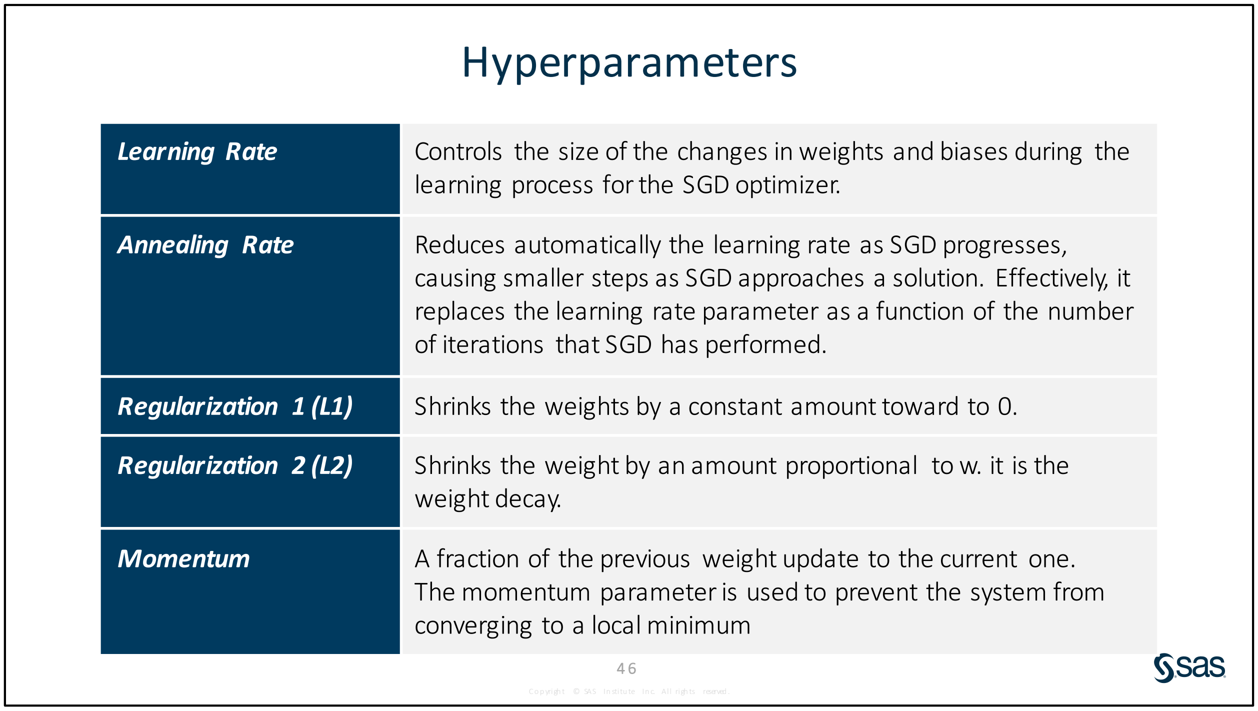
- Uses a single training observation to calculate an approximate gradient for each descent step
- Results in a chaotic progression to the error minima

Stochastic gradient descent (SGD) is another numerical optimization method available for the Neural Network task in Visual Data Mining and Machine Learning.

Stochastic gradient descent is a stochastic approximation of the gradient descent optimization. It approximates the true gradient by using a single data point in the training data set.

The gradient descent is an optimization algorithm to find the minimum value for a function iteratively. It takes steps proportional to the negative of the gradient of the function at the current point. The gradient is a multi-variable generalization of the derivative. The derivative can be defined on functions of a single variable. For functions of several variables, which is the case of the predictive models, including the neural networks, the gradient is defined. As the gradient represent the slope of the tangent for a particular function, it points in the direction of the greatest rate of increase of the function, which in the neural network case would be the point that minimize the loss function.

If a multi-variable function ***F(x)*** is differentiable in a neighborhood of a point ***a***, ***F(x)*** decreases fastest if it goes from a point ***a*** in the direction of the negative gradient of ***F*** at ***a***.



Hyperparameters

Learning Rate	Controls the size of the changes in weights and biases during the learning process for the SGD optimizer.
Annealing Rate	Reduces automatically the learning rate as SGD progresses, causing smaller steps as SGD approaches a solution. Effectively, it replaces the learning rate parameter as a function of the number of iterations that SGD has performed.
Regularization 1 (L1)	Shrinks the weights by a constant amount toward to 0.
Regularization 2 (L2)	Shrinks the weight by an amount proportional to w. it is the weight decay.
Momentum	A fraction of the previous weight update to the current one. The momentum parameter is used to prevent the system from converging to a local minimum

46

These sets of hyperparameters are discussed below.

Learning Rate

The *learning rate* is a training parameter that controls the size of weight and bias changes in learning of the training algorithm. Neural networks are often trained by weight decay methods. This means that at each iteration, we calculate the derivative of the loss function with respect to each weight and subtract it from that weight. However, by doing that, the weights can change too much in each iteration, making the weights to big and tending to overfit the model. One way to avoid that is to multiply each derivative by a small value, the *learning rate* described above, before they subtract it from its corresponding weight.

You can think of a loss function as a surface, where each direction that you can move in represents the value of a weight. Gradient descent is like taking leaps in the current direction of the slope, and the learning rate is like the length of the leap that you take.

The learning rate is defined in the context of optimization, and it is related to minimize the loss function, in this particular case, of a neural network. We define a loss function for a neural network, and the goal is to minimize this loss function. For this optimization problem, we use, for example, gradient descent or other variants of a weight decay, where the model parameters (weights and biases in the network) are updated in a way to decrease the loss function.

Mathematically, if the loss function is L(X, W, b), then our goal is to minimize L. For a convex optimization problem like this, we use derivatives of the loss function ∇L.

Therefore, the updates of the model weights and biases are $W = W - \eta \nabla L$.

This ∇ here is called the learning rate. It determines how quickly or how slowly you want to update the parameters. Usually, one can start with a large learning rate, and gradually decrease the learning rate as the training progresses.

Annealing Rate

Annealing is a way to automatically reduce the learning rate as SGD progresses, causing smaller steps as SGD approaches a solution.

Regularization L1 and L2

A standard way for regularization is to penalize the weights, preventing them from growing too large. It relies strongly on the implicit assumption that a model with small weights is somehow simpler than a network with large weights, and a model with large weights tends to over fit. The penalties try to keep the weights small, close to zero, or even zero. An alternative name in literature for weight penalties is *weight decay* because it forces the weights to decay toward zero.

L2 norm

Penalizes the square value of the weight (which explains also the 2 in the name). Tends to drive all the weights to smaller values.

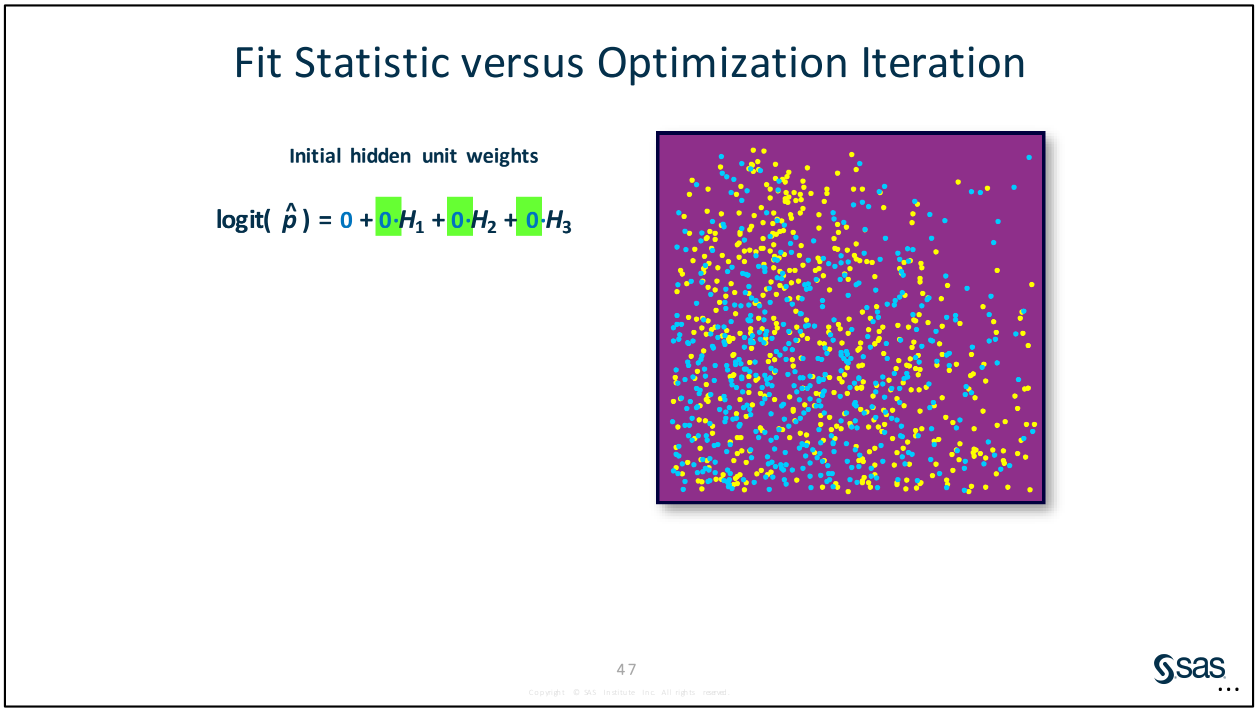
L1 norm

Penalizes the absolute value of the weight. Tends to drive some weights to exactly zero.

Momentum

Momentum simply adds a fraction *m* of the previous weight to update to the current one.

The momentum parameter is used to prevent the system from converging to a bad local minima.



Fit Statistic versus Optimization Iteration

Initial hidden unit weights

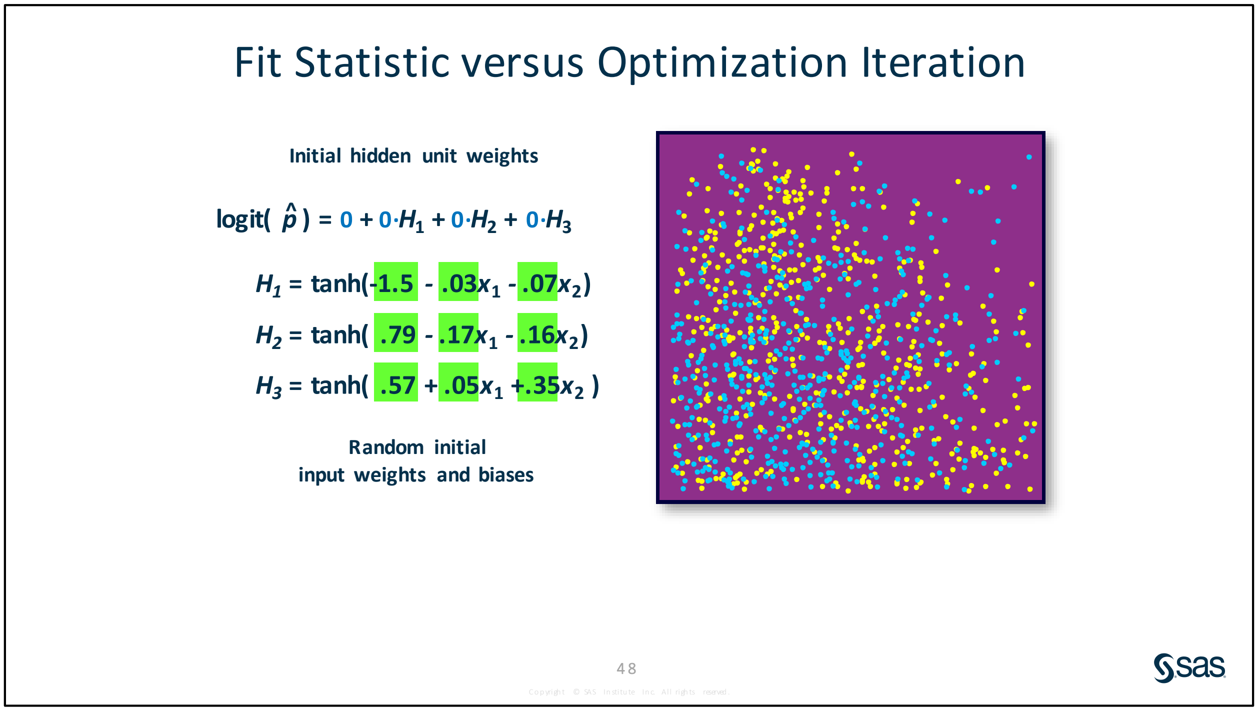
$\text{logit}(\hat{p}) = 0 + 0 \cdot H_1 + 0 \cdot H_2 + 0 \cdot H_3$

47

...

Complexity optimization is an integral part of neural network modeling. Other modeling methods select an optimal model from a sequence of possible models. In the Neural Network task, only one model is estimated, so what is compared?

SAS Visual Data Mining and Machine Learning treats each iteration in the optimization process as a separate model. The iteration with the smallest value of the selected fit statistic is chosen as the final model. This method of model optimization is called *stopped training*.



Fit Statistic versus Optimization Iteration

Initial hidden unit weights

$\text{logit}(\hat{p}) = 0 + 0 \cdot H_1 + 0 \cdot H_2 + 0 \cdot H_3$

$H_1 = \tanh(-1.5 - .03x_1 - .07x_2)$

$H_2 = \tanh(.79 - .17x_1 - .16x_2)$

$H_3 = \tanh(.57 + .05x_1 + .35x_2)$

Random initial
input weights and biases

48

To begin model optimization, model weights are given initial values. The weights multiplying the hidden units in the logit equation are set to zero, and the bias in the logit equation is set equal to the logit(π1), where π1 equals the primary outcome proportion. The remaining weights (corresponding to the hidden units) are given random initial values (near zero).

This “model” assigns each case a prediction estimate: $\hat{p}_i = \pi_1$. An initial fit statistic is calculated on training and validation data. For a binary target, this is proportional to the log likelihood function:

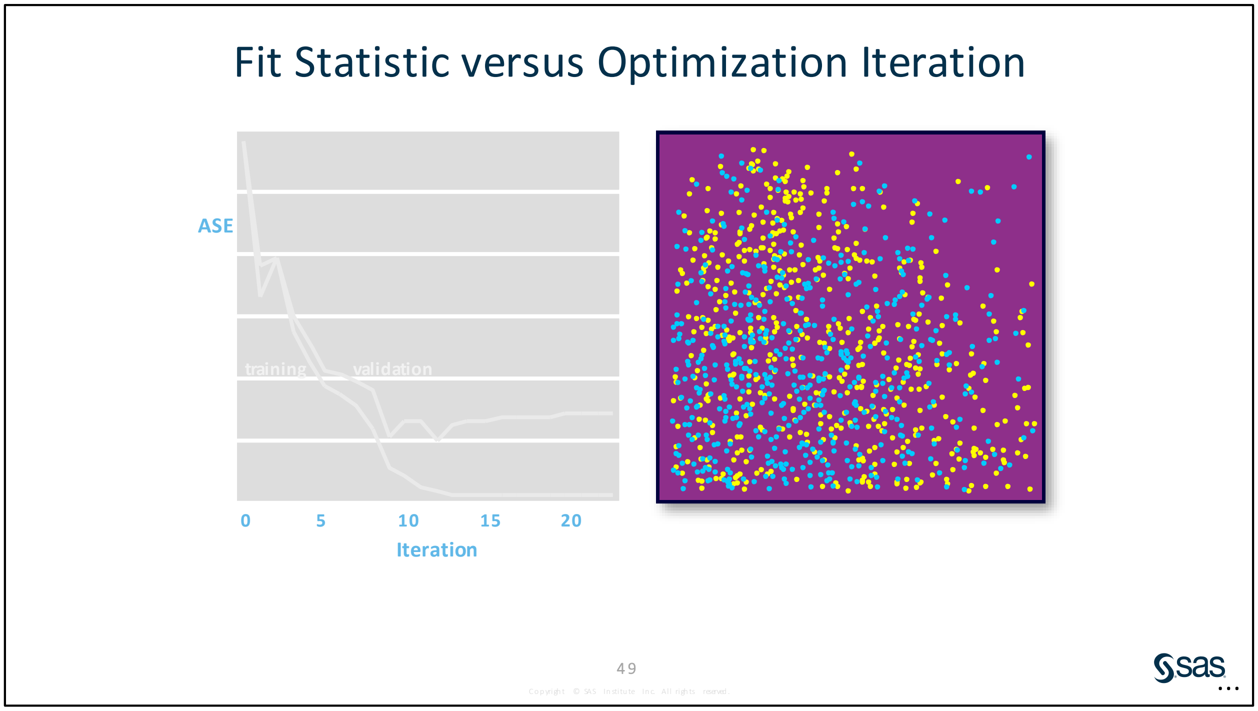
$$\sum_{\substack{primary\\outcomes}} \log(\hat{p}_i(\hat{\mathbf{w}})) + \sum_{\substack{secondary\\outcomes}} \log(1-\hat{p}_i(\hat{\mathbf{w}}))$$

where

$\hat{p}_i$ is the predicted target value.

$\hat{\mathbf{w}}$ is the current estimate of the model parameters.

Training proceeds by updating the parameter estimates in a manner that decreases the value of the objective function.



Fit Statistic versus Optimization Iteration

As stated above, in the initial step of the training procedure, the neural network model is set up to predict the overall average response rate for all cases.

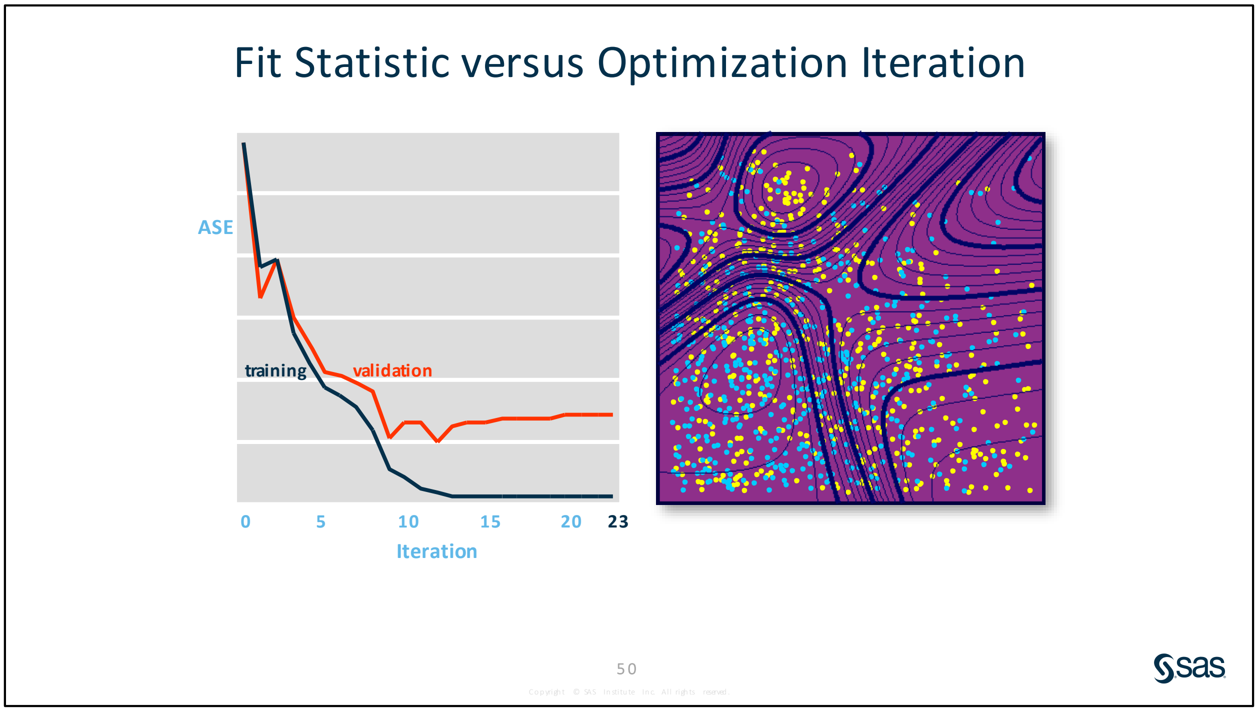
One step substantially decreases the value average squared error (ASE). Amazingly, the model that corresponds to this one-iteration neural network closely resembles the standard regression model, as seen from the fitted isoclines.

The second iteration step goes slightly astray. The model actually becomes slightly worse on the training and validation data.

Things are back on track in the third iteration step. The fitted model is already exhibiting nonlinear and nonadditive predictions. Half of the improvement in ASE is realized by the third step.

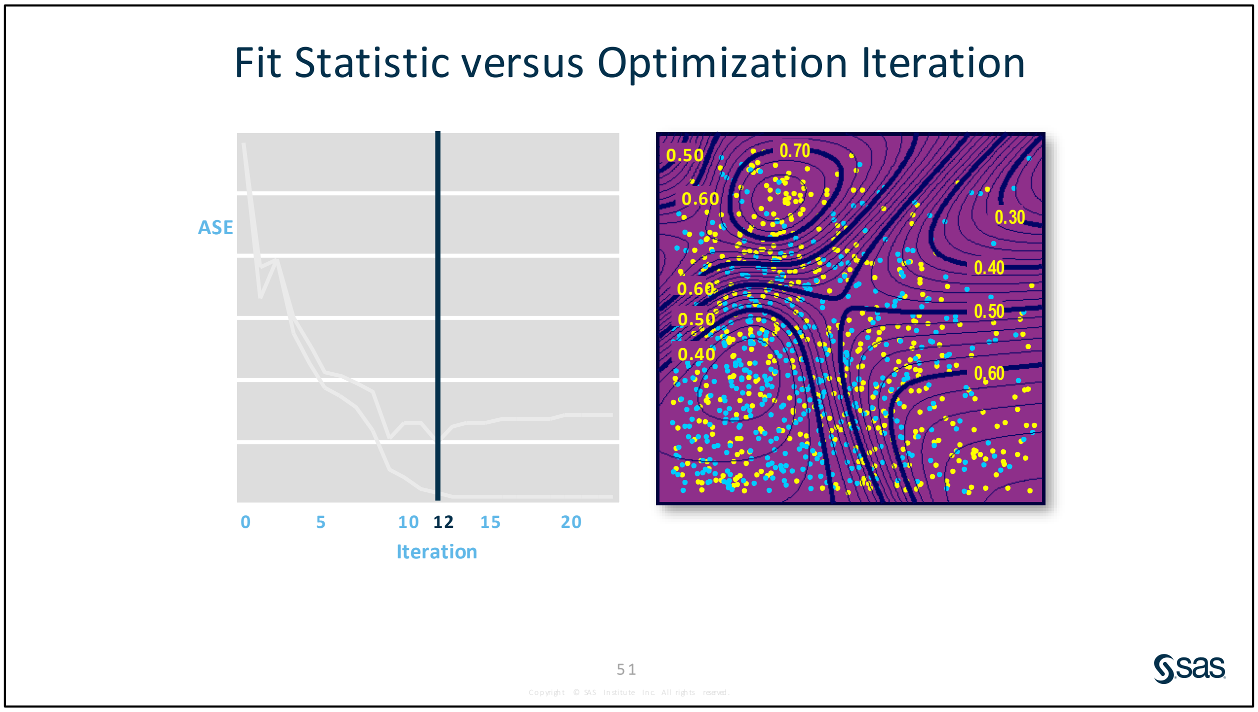
Most of the improvement in validation ASE occurred by the ninth step. (Training ASE continues to improve until convergence in step 23.) The predictions are close to their final form.

Step 12 brings the minimum value for validation ASE. Although this model is ultimately chosen as the final model, SAS Enterprise Miner continues to train until the likelihood objective function changes by a negligible amount on the training data.



Fit Statistic versus Optimization Iteration

In step 23, training is declared complete due to lack of change in the objective function from step 22. Notice that between step 13 and step 23, ASE actually increased for the validation data. This is an indication of overfitting.

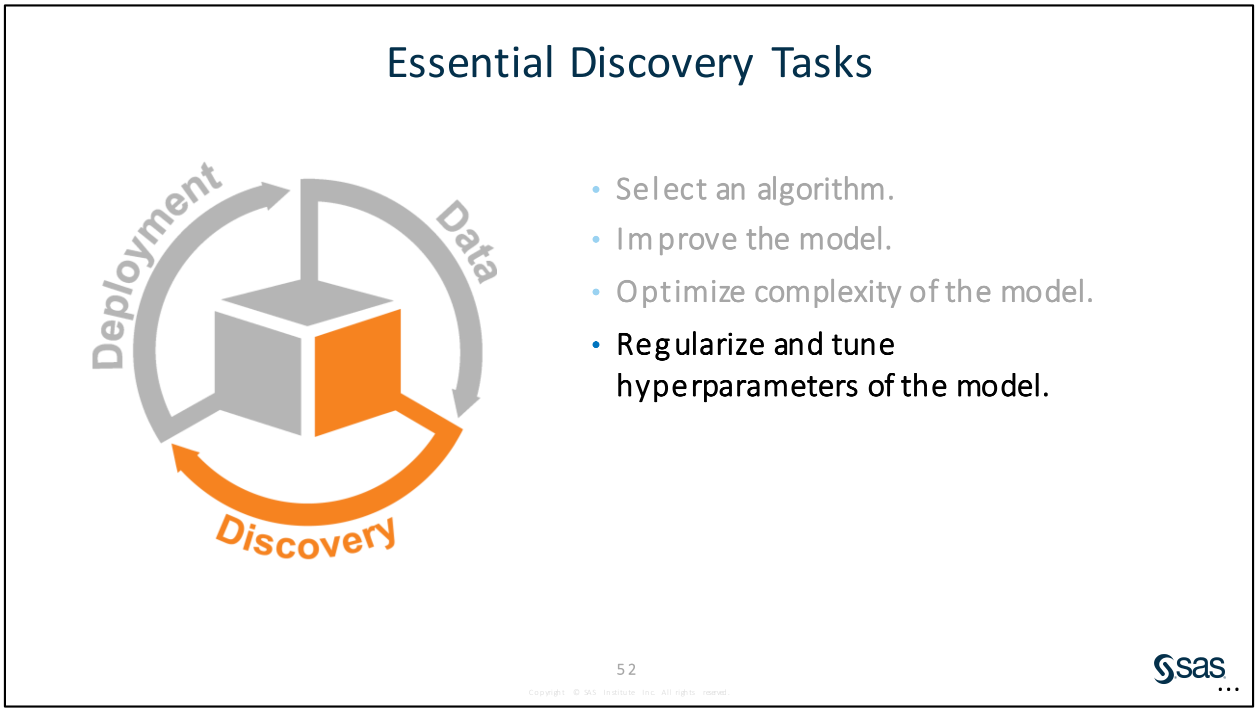


Fit Statistic versus Optimization Iteration

51

The Neural Network task selects the modeling weights from iteration 13 for the final model. In this iteration, the validation ASE is minimized. You can also configure the Neural Network task to select the iteration with minimum misclassification for final weight estimates.

Note: The name *stopped training* comes from the fact that the final model is selected as if training were stopped on the optimal iteration. Detecting when this optimal iteration occurs (while actually training) is somewhat problematic. To avoid stopping too early, the Neural Network task continues to train until convergence on the training data or until reaching the maximum iteration count, whichever comes first.

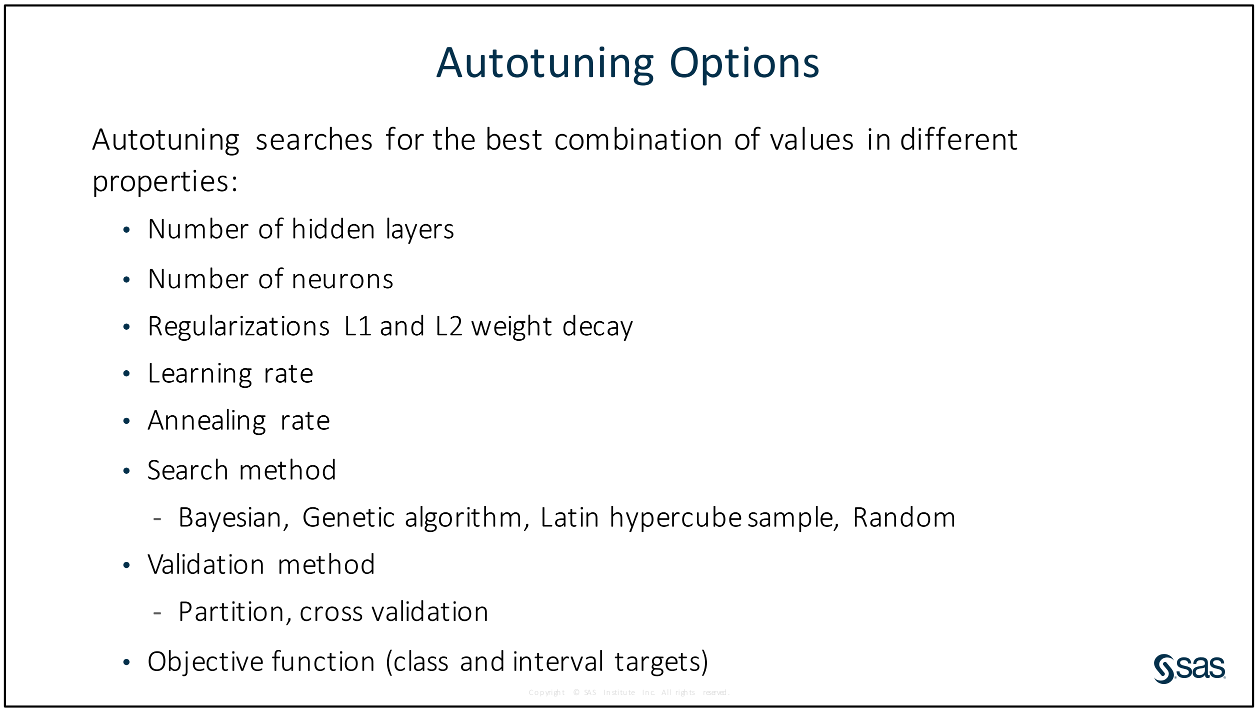


Essential Discovery Tasks

- Select an algorithm.
- Improve the model.
- Optimize complexity of the model.
- Regularize and tune hyperparameters of the model.

52

One of the hardest components in neural network modeling is finding the model parameters that minimize the loss function. These parameters are associated with the number of hidden layers, the number of hidden units, the activation function, the target function, and so on. As the complexity of your model increases, its predictive abilities often decrease after a certain point due to overfitting and multicollinearity issues. Therefore, the resulting models often do not generalize well to new data, and they yield unstable parameter estimates. Some of the machine learning procedures in SAS Viya offer the Autotune option, which searches the optimal combination of hyperparameters to fit the best model under certain conditions.



Autotuning Options

Autotuning searches for the best combination of values in different properties:

- Number of hidden layers
- Number of neurons
- Regularizations L1 and L2 weight decay
- Learning rate
- Annealing rate
- Search method
  - Bayesian, Genetic algorithm, Latin hypercube sample, Random
- Validation method
  - Partition, cross validation
- Objective function (class and interval targets)

When the Autotune feature is used, SAS Visual Data Mining and Machine Learning returns the optimal number of units for each hidden layer. Autotuning is invoked by selecting the Performing Autotuning option property on the Options tab in the Neural Network node.

Autotuning is available only when the number of hidden layers is less than 6.

The Autotuning statement activates the tuning optimization algorithm, which searches for the best hidden layers and regularization parameters based on the problem and specified options. If the algorithm used to train the neural network is based on the Stochastic Gradient Descent, the Autotune feature also searches for the best values of the learning rate and annealing rate. In addition, the Autotune feature searches for the best hyperparameter values for the number of hidden layers, the number of hidden units in each hidden layer, the L1 regularization, and the L2 regularization parameters.

You can also define the search method for the hyperparameters, as Bayesian, Genetic algorithm, Latin hypercube sample, or Random sample. The genetic algorithm method uses an initial Latin hypercube sample that seeds a genetic algorithm to generate a new population of alternative configurations at each iteration. The Latin Hypercube method performs an optimized grid search that is uniform in each tuning parameter, but random in combinations. The Random method generates a single sample of purely random configurations. The Bayesian method uses priors to seed the iterative optimization.

Finally, you can specify the number of tuning evaluations in one iteration. This option is available only if the Search method is Genetic algorithm or Bayesian. Similarly, you can specify the maximum number of tuning evaluations and the maximum number of tuning iterations.

For the search method, Random or Latin hypercube is also possible to specify a sample size.

Finally, you can specify the validation method for finding the objective value, including partition and cross validation – including the proportion of the validation data set and the number of folds for cross validation – and the objective function depending on the level of the target variable.

Autotuning searches for the best combination of the neural network parameters. ***Performing autotuning can substantially increase run time.***

Autotuning runs based on some options, which limit the search of all possible combinations in terms of the neural network parameters.

Number of Hidden Layers specifies whether to autotune the number of hidden layers. It ranges from 1 to 5. The default initial value is 1. The default range is from 0 to 2.

Number of Neurons specifies whether to autotune the number of neurons. It ranges from 1 to 1000. The default initial value is 1. The default range is from 1 to 100.

L1 Weight Decay specifies whether to autotune the L1 weight decay parameter. It penalizes the absolute value for the weights. Different values of L1 are tried between the range defined by From and To. The default initial value for the L1 is 0. The default range is from 0 to 10.

L2 Weight Decay specifies whether to autotune the L2 weight decay parameter. It penalizes the square value for the weights. Different values of L2 are tried between the range established by from and to. The default initial value for the L2 is 0. The default range is from 0 to 10.

Learning Rate specifies whether to autotune the learning rate for the hidden layers. It controls the size of the weight changes. It ranges from 0 (exclusive) to 1. The default initial value is 0.1. The default initial value for the learning rate is 0.1. The default for the range is from 0.01 to 1. It works just for the SGD algorithm.

Annealing Rate specifies whether to autotune the annealing rate for the hidden layers. It automatically reduces the learning rate as SGD progresses. The default initial value is 0.001. The default range is from 0.000001 to 0.1. It works just for the SGD algorithm.

Search Options specifies the options for autotuning searching. The following options are available:

- **Genetic algorithm** uses an initial Latin hypercube sample that seeds a genetic algorithm. The genetic algorithm generates a new population of alternative configurations at each iteration.
- **Latin hypercube sample** performs an optimized grid search that is uniform in each tuning parameter, but random in combinations.
- **Random** generates a single sample of purely random configurations.
- **Bayesian** uses priors to seed the iterative optimization.

Number of evaluations per iteration specifies the number of tuning evaluations in one iteration. This option is available only if the Search method is Genetic algorithm or Bayesian. The default value is 10. It ranges from 2 to 2,147,483,647.

Maximum number of evaluations specifies the maximum number of tuning evaluations. This option is available only if the Search method is Genetic algorithm or Bayesian. The default value is 50. It ranges from 3 to 2,147,483,647.

Maximum number of iterations specifies the maximum number of tuning iterations. This option is available only if the Search method is Genetic algorithm or Bayesian. The default value is 5. It ranges from 1 to 2,147,483,647.

Sample size specifies the sample size. This option is available only if the Search method is Random or Latin Hypercube sample. The default value is 50. It ranges from 2 to 2,147,483,647.

There are some general options associated with the autotuning search.

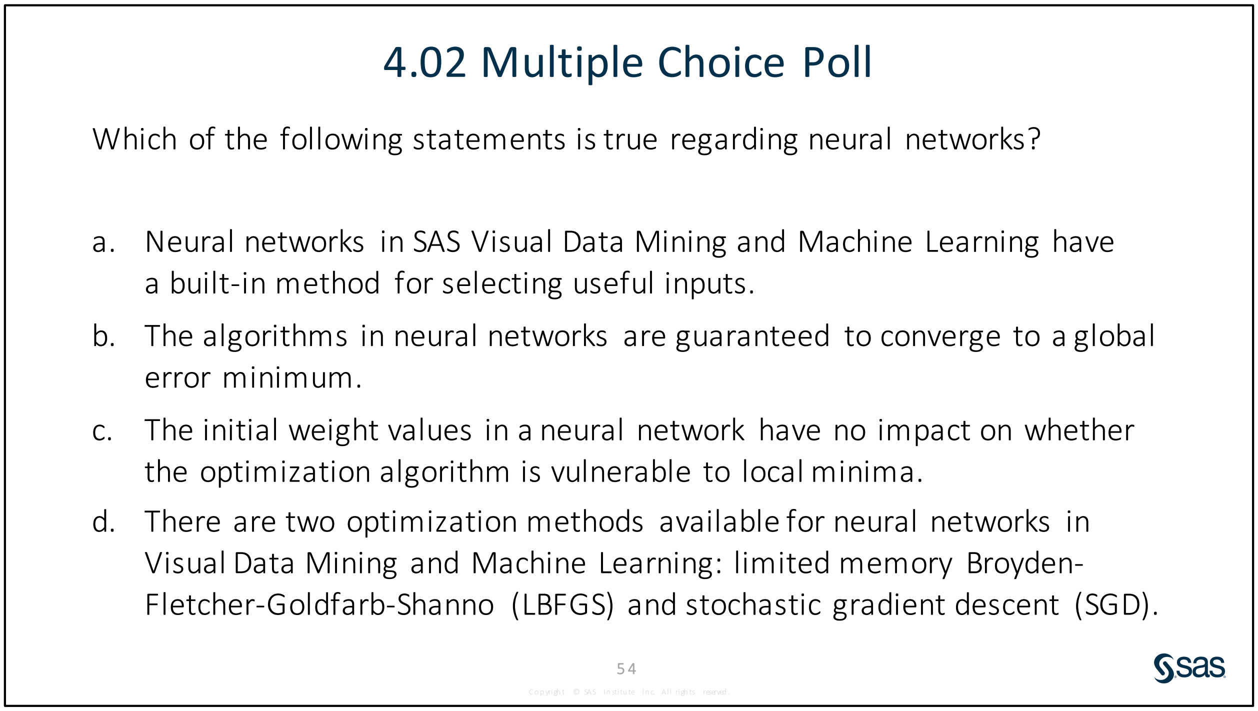
Validation method specifies the validation method for finding the objective value. If your data is partitioned, then that partition is used. Validation method, Validation data proportion, and Cross validation number of folds are all ignored.

- **Partition** specifies using the partition validation method. With partition, you specify proportions to use for randomly assigning observations to each role.
  - **Validation data proportion** specifies the proportion of data to be used for the Partition validation method. The default value is 0.3.

- **K-fold cross validation** specifies using the cross validation method. In cross validation, each model evaluation requires k training executions (on k-1 data folds) and k scoring executions (on one holdout fold). This increases the evaluation time by approximately a factor of k.
  - **Cross validation number of folds** specifies the number of partition folds in the cross validation process (the k defined above). Possible values range from 2 to 20. The default value is 5.

Nominal target objective function specifies the objective function to optimize for tuning parameters for a nominal target. Possible values are average squared error, area under the curve, F1 score, F0.5 score, gamma, Gini coefficient, Kolmogorov-Smirnov statistic, multi-class log loss, misclassification rate, root average squared error, and Tau. The default value is misclassification rate.

Interval target objective function specifies the objective function to optimize for tuning parameters for an interval target. Possible values are average squared error, mean absolute error, mean squared logarithmic error, root average squared error, root mean absolute error, and root mean squared logarithmic error. The default value is average squared error.



4.02 Multiple Choice Poll

Which of the following statements is true regarding neural networks?

a. Neural networks in SAS Visual Data Mining and Machine Learning have a built-in method for selecting useful inputs.

b. The algorithms in neural networks are guaranteed to converge to a global error minimum.

c. The initial weight values in a neural network have no impact on whether the optimization algorithm is vulnerable to local minima.

d. There are two optimization methods available for neural networks in Visual Data Mining and Machine Learning: limited memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) and stochastic gradient descent (SGD).

54

Improving a Neural Network Model by Changing the Network Learning and Optimization Parameters

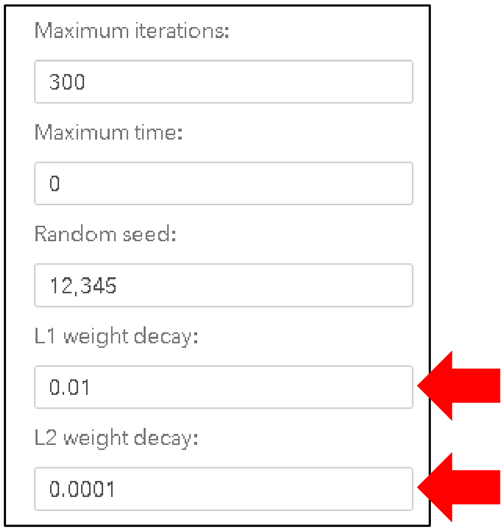
In this demonstration, you change the previous settings of the Neural Network node in the Chapter 4 pipeline. You modify the learning and optimization parameters and compare this model performance to the other model in the pipeline.

1. Recall that in the previous model, based on changes in the network architecture, achieved an average square error of 0.0697 on the VALIDATE partition. This fit statistic was better than the first model by using the default settings, approximately 6%.

   Try to improve the neural network performance by changing now some of the default settings assigned to the learning and optimization parameters.

2. Under the **Common Optimization Options** properties, increase **L1 weight decay** from **0** to **0.01**.

3. Decrease **L2 weight decay** from **0.1** to **0.0001**.



Maximum iterations:

300

Maximum time:

0

Random seed:

12,345

L1 weight decay:

0.01

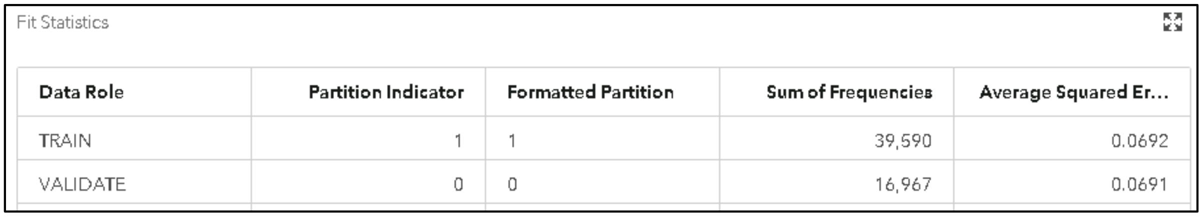
L2 weight decay:

0.0001

4. Run the Neural Network node.

5. Open the results for the Neural Network node.

6. Click the **Assessment** tab.



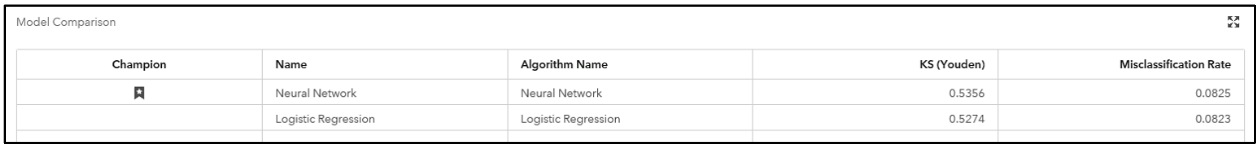
Fit Statistics

Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.0692
VALIDATE	0	0	16,967	0.0691

   The average square error for the tuned Neural Network model is 0.0691 on the VALIDATE partition. This fit statistic is slightly better than the previous model, approximately 1%.

7. Close the Results window.

8. Run the entire pipeline and view the results of model comparison. The Neural Network model is the champion of the pipeline based on default KS.



Model Comparison

Champion	Name	Algorithm Name	KS (Youden)	Misclassification Rate
🔖	Neural Network	Neural Network	0.5356	0.0825
	Logistic Regression	Logistic Regression	0.5274	0.0823

9. Close the Results window.

End of Demonstration

Exercises

1. Building a Neural Network

a. Build a neural network using the Autotune feature. Add a Neural Network node to the Chapter 4 pipeline, below the Variable Selection node. Use the Autotune feature. Explore the settings that are made available when **Autotune** is selected. Run a few options by changing the range of the parameters search.

Note: This exercise might take several minutes to run.

b. What criteria were selected for the champion model?

- Number of hidden layers
- Number of hidden nodes
- Architecture
- Optimization technique

c. How does the autotuned neural network compare to the other models in the pipeline, particularly to the Neural Network model built during the demonstration? Consider the fit statistic average square error for this comparison.

End of Exercises

4.4 Solutions

Solutions to Exercises

1. Building a Neural Network

a. Build a neural network using the Autotune feature. Add a Neural Network node to the Chapter 4 pipeline, below to the Variable Selection node. Use the Autotune feature. Explore the settings that turn on when **Autotune** is selected, but keep all properties at their defaults.

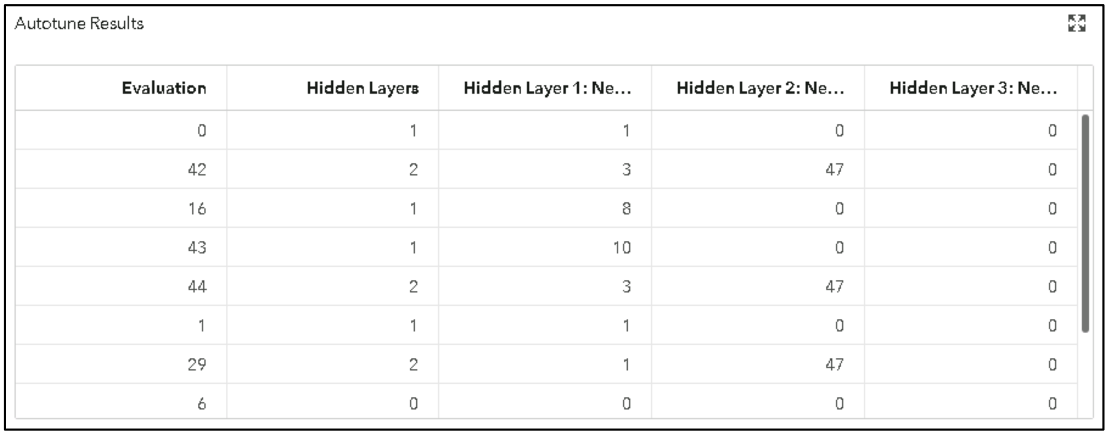
1) On the Starter Template pipeline, right-click the **Variable Selection** node and select **Add below ⇨ Supervised Learning ⇨ Neural Network**.

2) In the properties pane, turn on the **Perform Autotuning** option. The default properties show starting values and ranges that are tried for each property in the Neural Network model.

3) Right-click the **Neural Network** node and select **Run**. This process might take few minutes.

4) When the execution is over, right-click the **Neural Network** node and select **Results**.

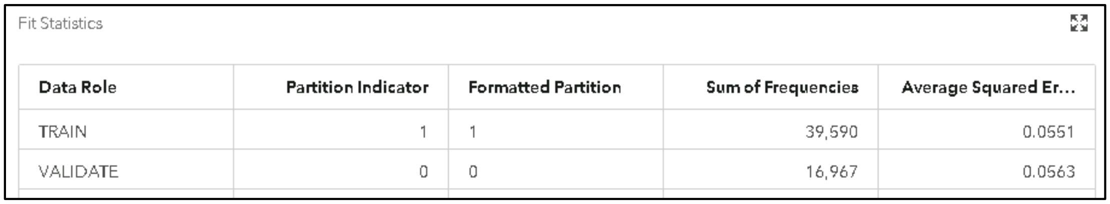
5) Examine the Results window. Maximize the Autotune Results window and notice the different evaluations performed. Restore the Autotune Results window.



Autotune Results

Evaluation	Hidden Layers	Hidden Layer 1: Ne...	Hidden Layer 2: Ne...	Hidden Layer 3: Ne...
0	1	1	0	0
42	2	3	47	0
16	1	8	0	0
43	1	10	0	0
44	2	3	47	0
1	1	1	0	0
29	2	1	47	0
6	0	0	0	0

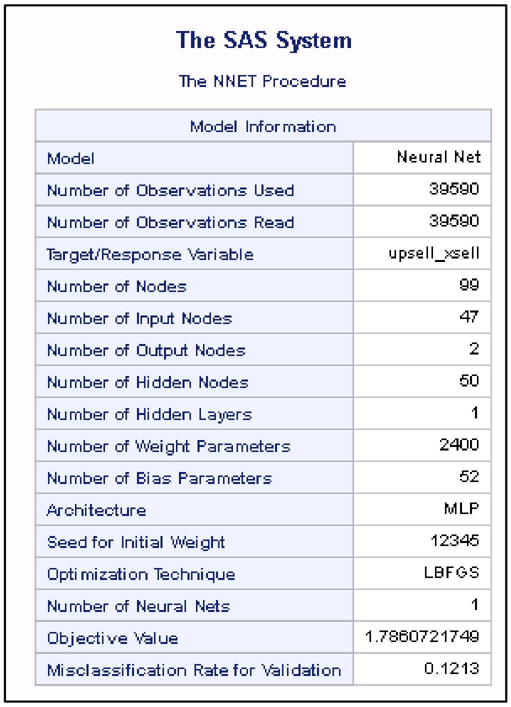
6) Scroll down and observe the Fit Statistics window. The average square error for the Autotune model is 0.0563 on the VALIDATE partition.



Fit Statistics

Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.0551
VALIDATE	0	0	16,967	0.0563

7) Scroll down and maximize the Output window. This output shows the set of parameters selected for the final Neural Network model.



The SAS System

The NNET Procedure

Model Information	
Model	Neural Net
Number of Observations Used	39590
Number of Observations Read	39590
Target/Response Variable	upsell_xsell
Number of Nodes	99
Number of Input Nodes	47
Number of Output Nodes	2
Number of Hidden Nodes	50
Number of Hidden Layers	1
Number of Weight Parameters	2400
Number of Bias Parameters	52
Architecture	MLP
Seed for Initial Weight	12345
Optimization Technique	LBFGS
Number of Neural Nets	1
Objective Value	1.7860721749
Misclassification Rate for Validation	0.1213

b. What criteria were selected for the champion model?

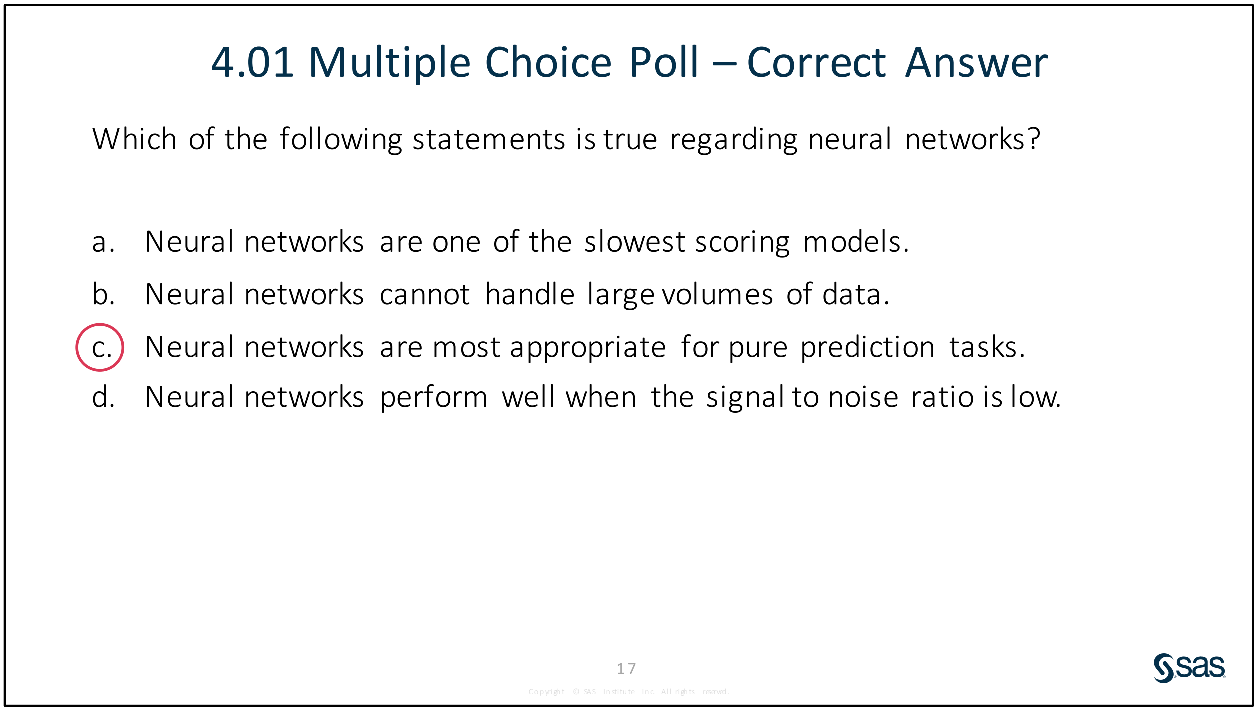
- Number of hidden layers: **2**
- Number of hidden nodes: **50**
- Architecture: **MLP**
- Optimization technique: **SGD**

c. How does the autotuned neural network compare to the other models in the pipeline, particularly to the Neural Network model built during the demonstration? Consider the fit statistics average square error for this comparison.

The model's performance is worse than the demonstration's model performance.

End of Solutions

Solutions to Student Activities (Polls/Quizzes)



4.01 Multiple Choice Poll – Correct Answer

Which of the following statements is true regarding neural networks?

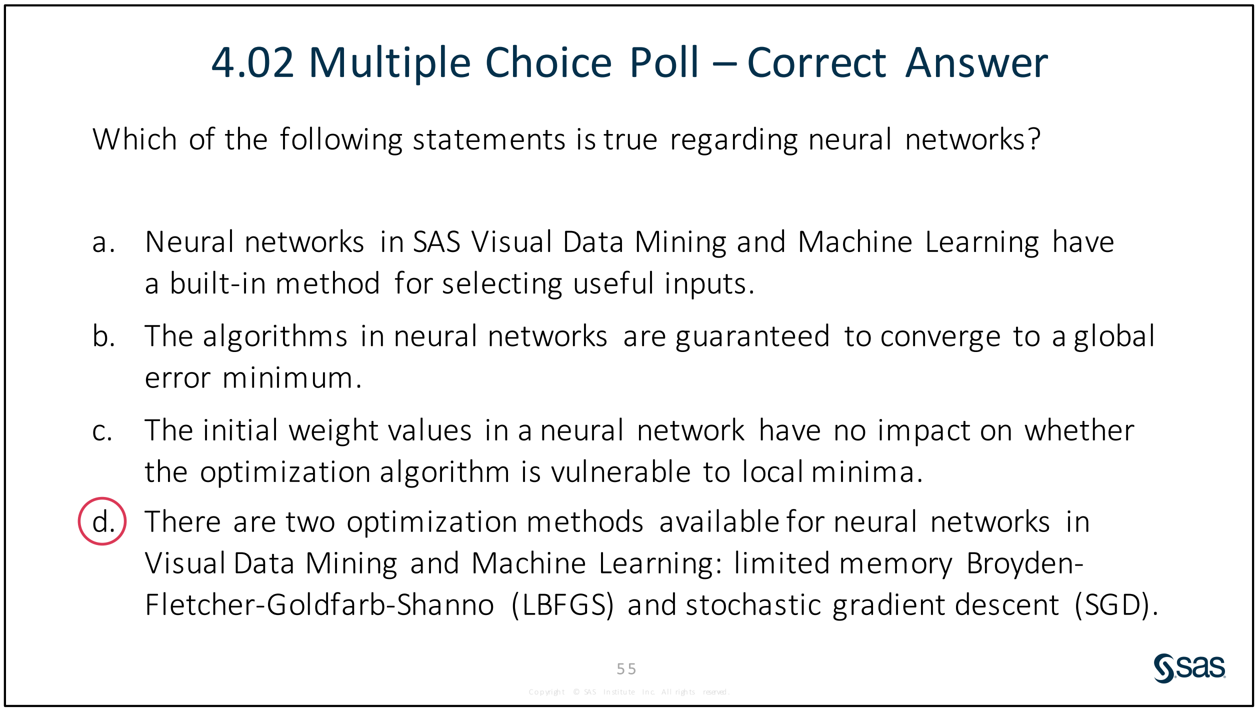
a. Neural networks are one of the slowest scoring models.

b. Neural networks cannot handle large volumes of data.

[✓ c.] Neural networks are most appropriate for pure prediction tasks.

d. Neural networks perform well when the signal to noise ratio is low.

17



4.02 Multiple Choice Poll – Correct Answer

Which of the following statements is true regarding neural networks?

a. Neural networks in SAS Visual Data Mining and Machine Learning have a built-in method for selecting useful inputs.

b. The algorithms in neural networks are guaranteed to converge to a global error minimum.

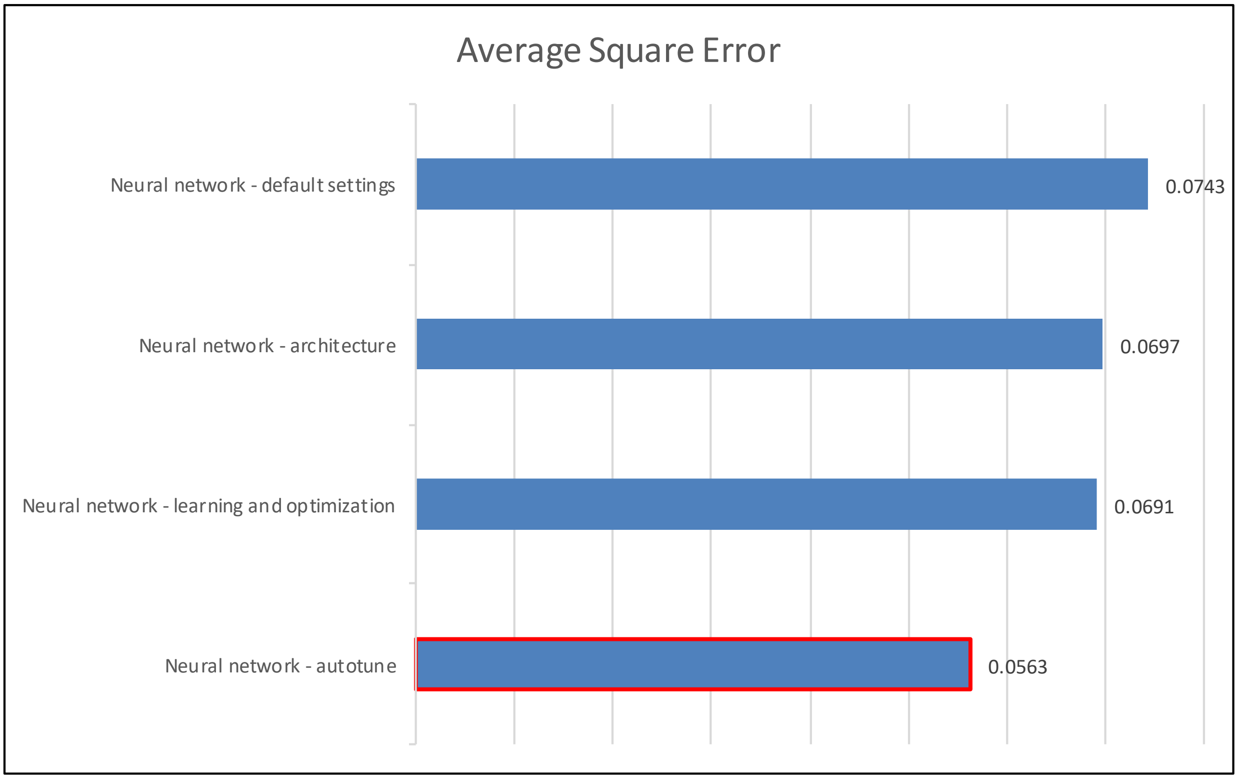
c. The initial weight values in a neural network have no impact on whether the optimization algorithm is vulnerable to local minima.

[✓ d.] There are two optimization methods available for neural networks in Visual Data Mining and Machine Learning: limited memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) and stochastic gradient descent (SGD).

55

Summary of Neural Network Models

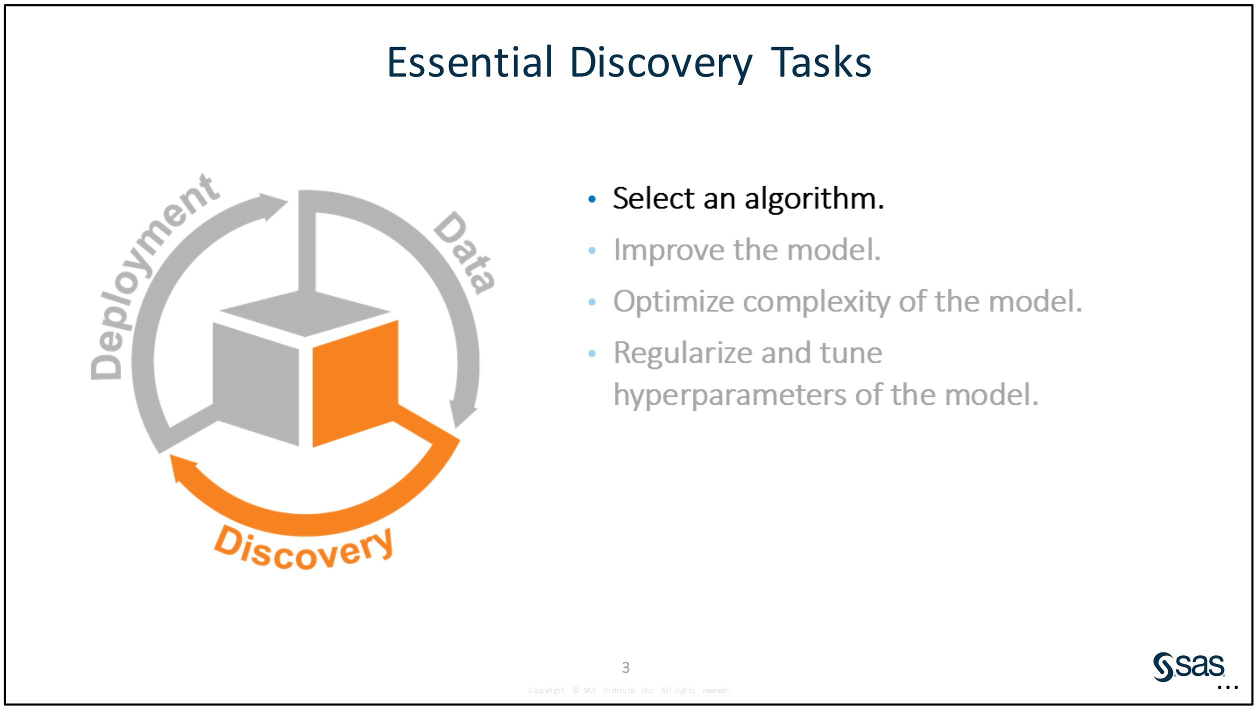
The following chart shows the different performances for the Neural Network models. The Neural Network model was improved based on the topics covered in this chapter, such as the introduction, the network architecture, and the different options for learning and optimization. Starting from the default settings, each one of those topics were applied to improve the model's performance. Finally, the neural network based on the Autotune feature was created.



The Autotune feature achieved the best model, beating the neural network with the default settings and the ones with the tuning on the parameters based on the network architecture and the learning and optimization options.

Chapter 5 Support Vector Machines and Additional Topics

5.1 Large-Margin Linear Classifier

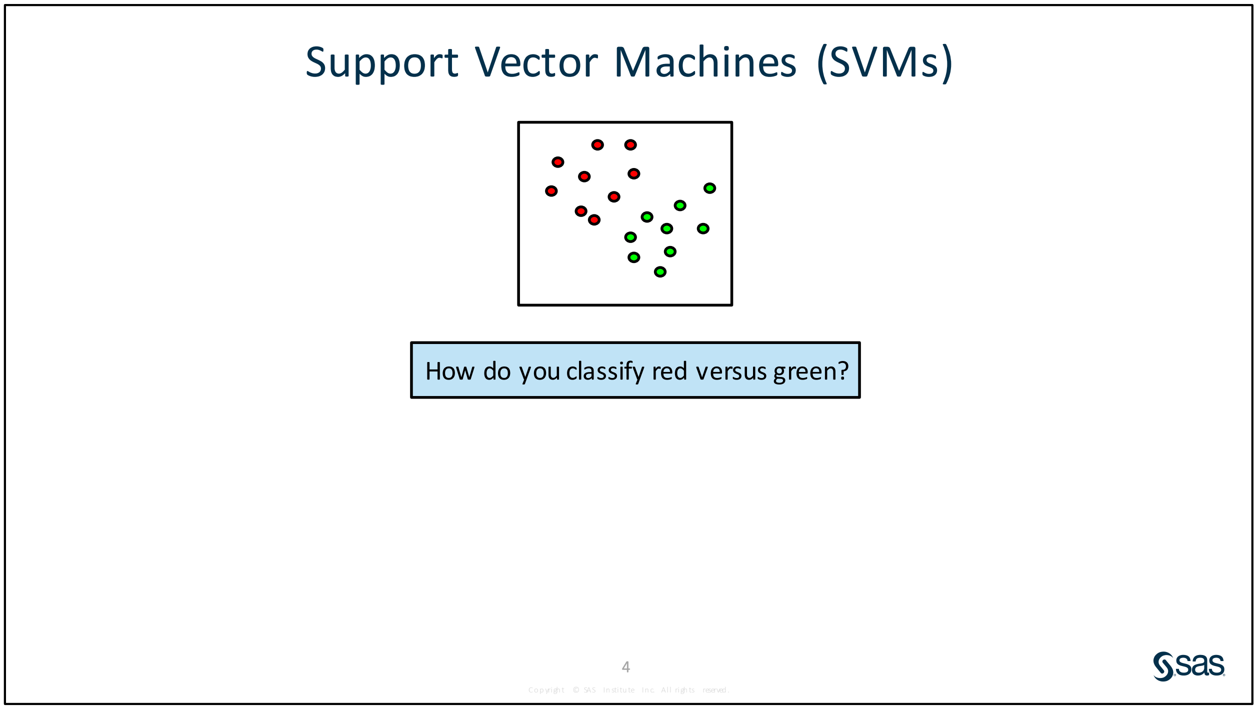


Essential Discovery Tasks

Deployment
Data
Discovery

- Select an algorithm.
- Improve the model.
- Optimize complexity of the model.
- Regularize and tune hyperparameters of the model.

Support vector machines are another type of supervised machine learning algorithm. Therefore, they apply to the Model component under the Discovery phase of the analytics life cycle.

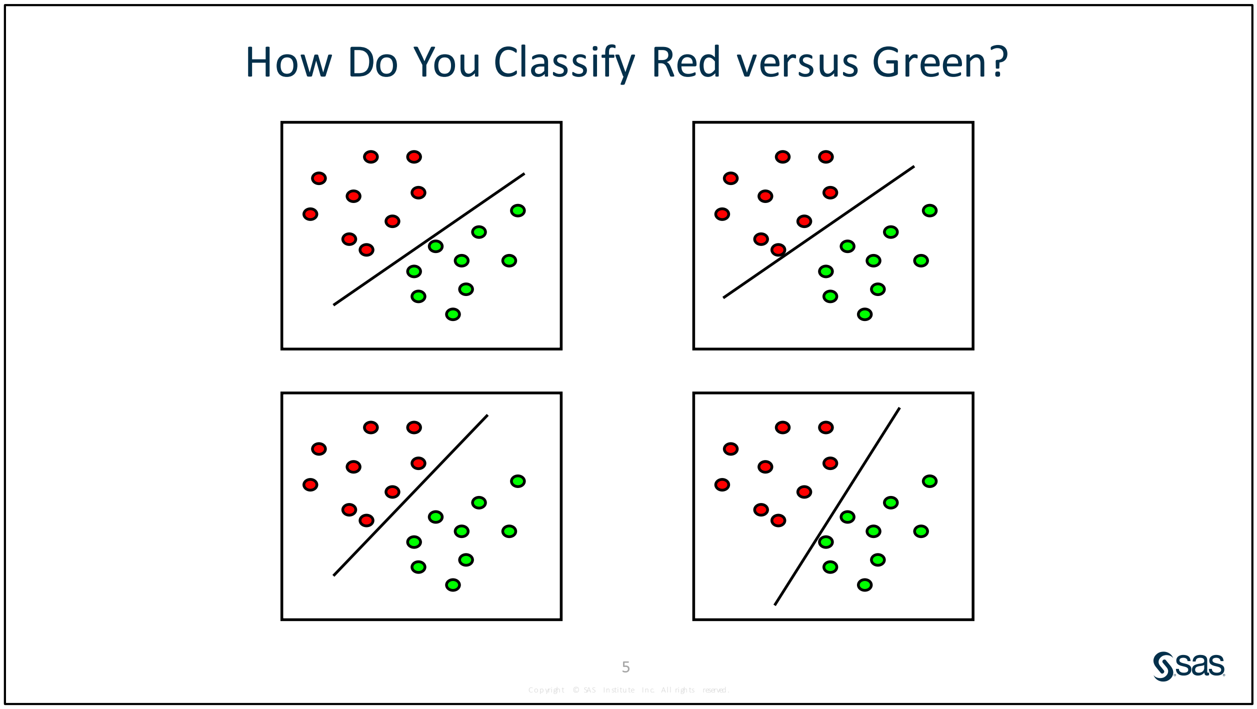


Support Vector Machines (SVMs)

How do you classify red versus green?

Support vector machine (SVMs) were originally developed for pure classification tasks to solve pattern recognition problems. They have since expanded and now can be used for regression tasks as well (Vapnik, Golowich, and Smola). However, in Model Studio, currently only classification tasks are possible for binary targets.

Support vector machines have been broadly used in fields such as image classification, handwriting recognition, financial decision, and text mining.

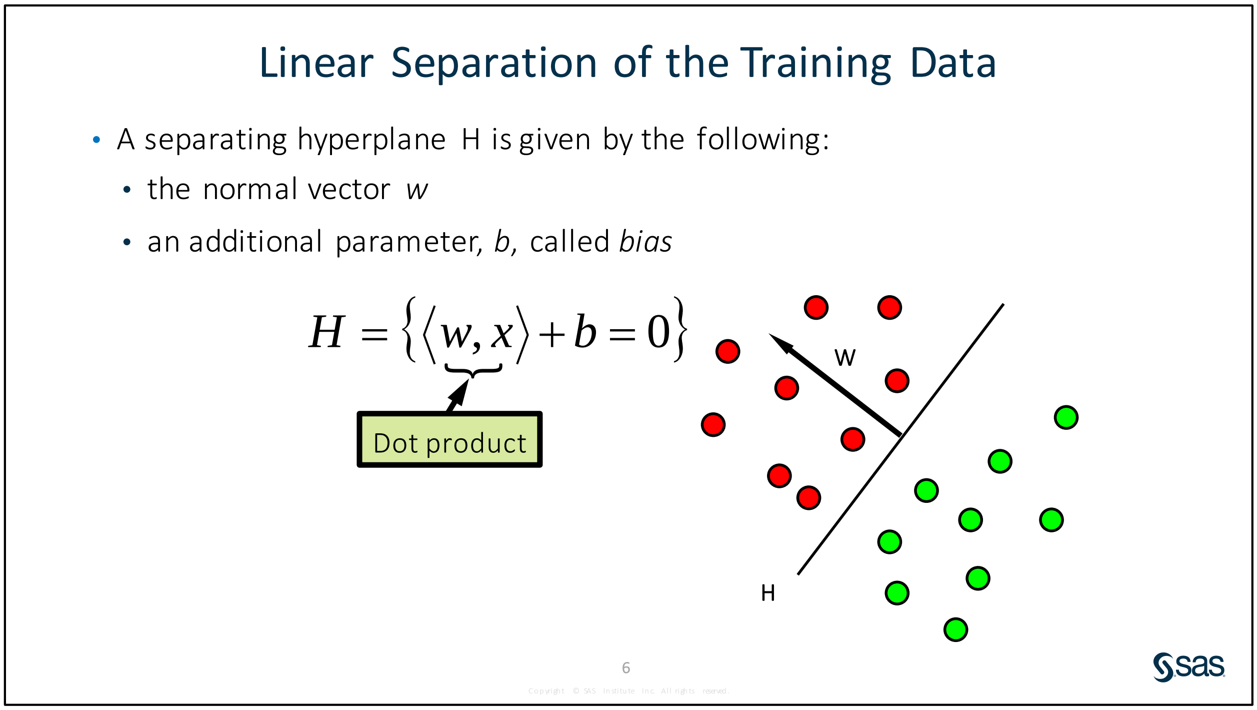


How Do You Classify Red versus Green?

5

In this simple illustration, the goal is to classify red (the dots in the upper left on the slide above) versus green (in the lower right). There are many classification rules (lines) that could be used to perfectly separate the red and green cases. In fact, when data are perfectly linearly separable, as is the case above, there are infinitely many solutions. So how will a unique solution be discovered?

Given two input variables (as shown above), the support vector machine is a line. Given three input variables, the support vector is a plane, and with more than three input variables, the support vector is a hyperplane.



Linear Separation of the Training Data

- A separating hyperplane H is given by the following:
  - the normal vector *w*
  - an additional parameter, *b,* called *bias*

$$H = \{\underbrace{\langle w, x\rangle}_{\text{Dot product}} + b = 0\}$$

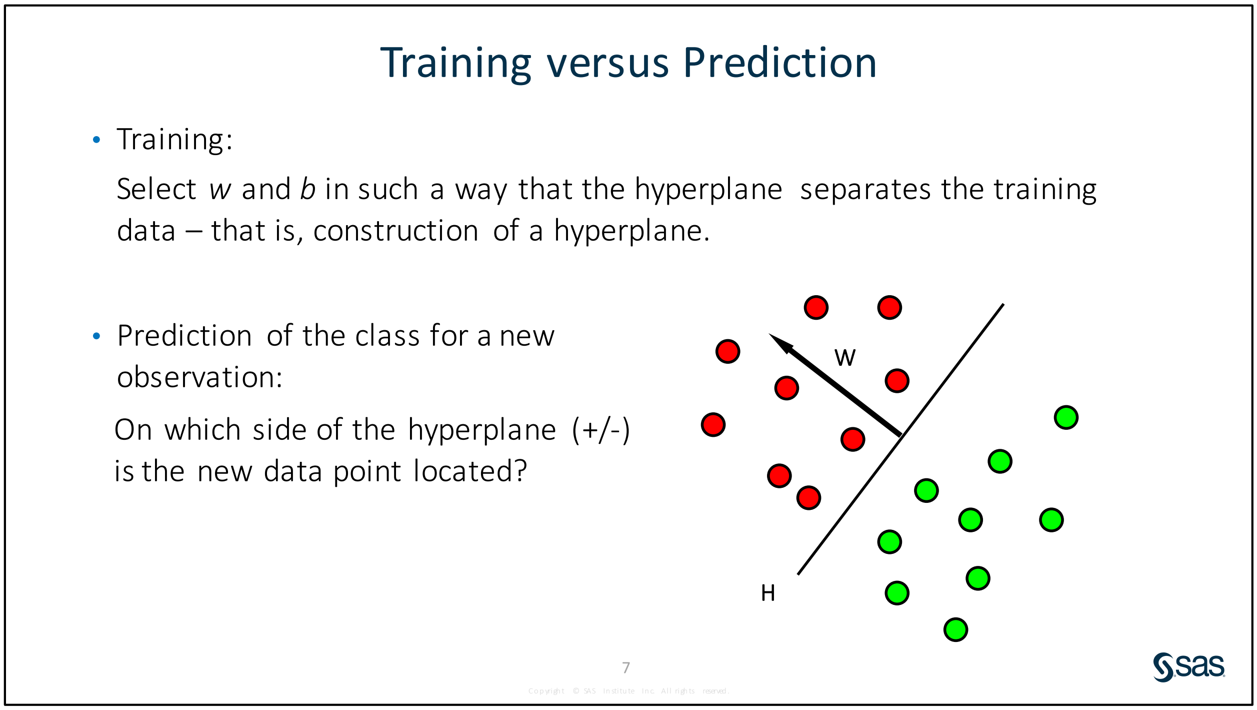
6

For mathematical convenience, the binary target is defined by values +1 and -1, rather than the usual 1 and 0. The renumbering is done automatically by Model Studio. Because the linear separator equals 0, classification is determined by a point falling on the positive or negative side of the line.

This is a simple linear problem to start with. Later, you see a more complex nonlinear problem. In the illustration above, think of the vector *w* as the mechanism that affects the slope of *H*. The bias parameter, *b*, is the measure of offset of the separating line (or plane, in higher dimensions) from the origin.

A dot product is a way to multiply vectors that result in a scalar, or a single number, as the answer. It is an element by element multiplication, and then a sum across the products. Consider the following example:

If $\underline{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$ and $\underline{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$ then $\langle \underline{a}, \underline{b} \rangle = a_1b_1 + a_2b_2 + a_3b_3$.



Training versus Prediction

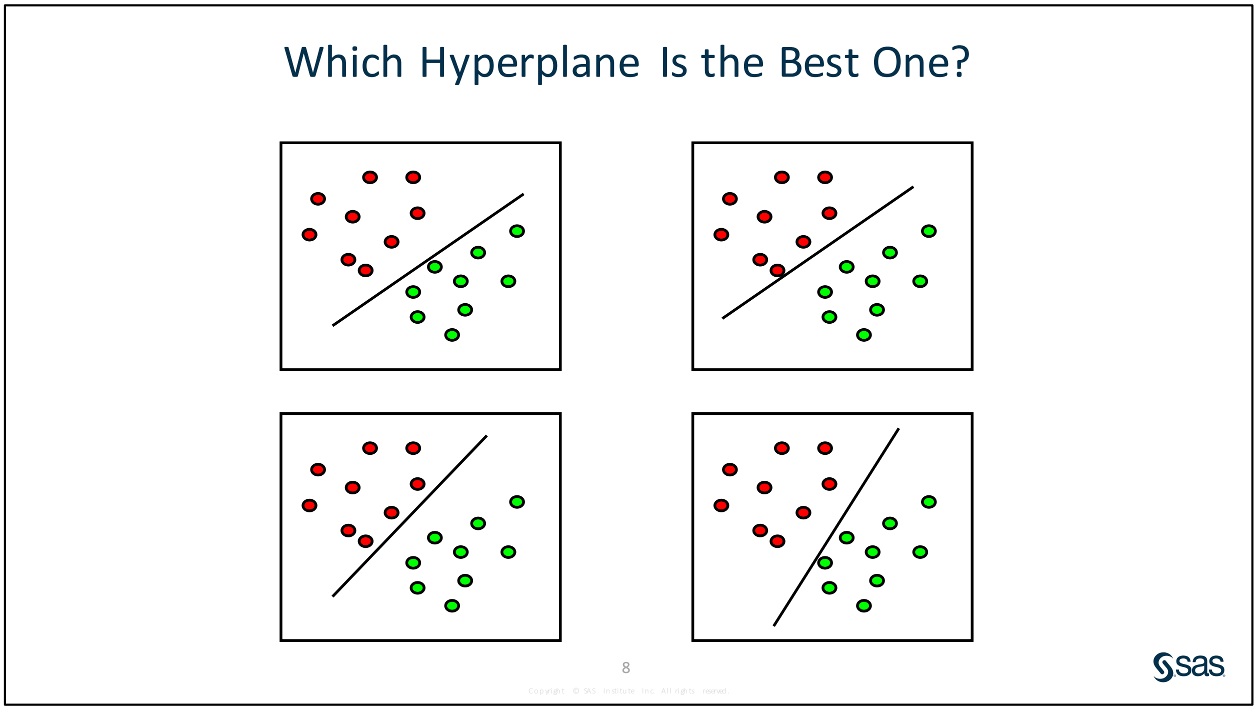
- Training:

  Select *w* and *b* in such a way that the hyperplane separates the training data – that is, construction of a hyperplane.

- Prediction of the class for a new observation:

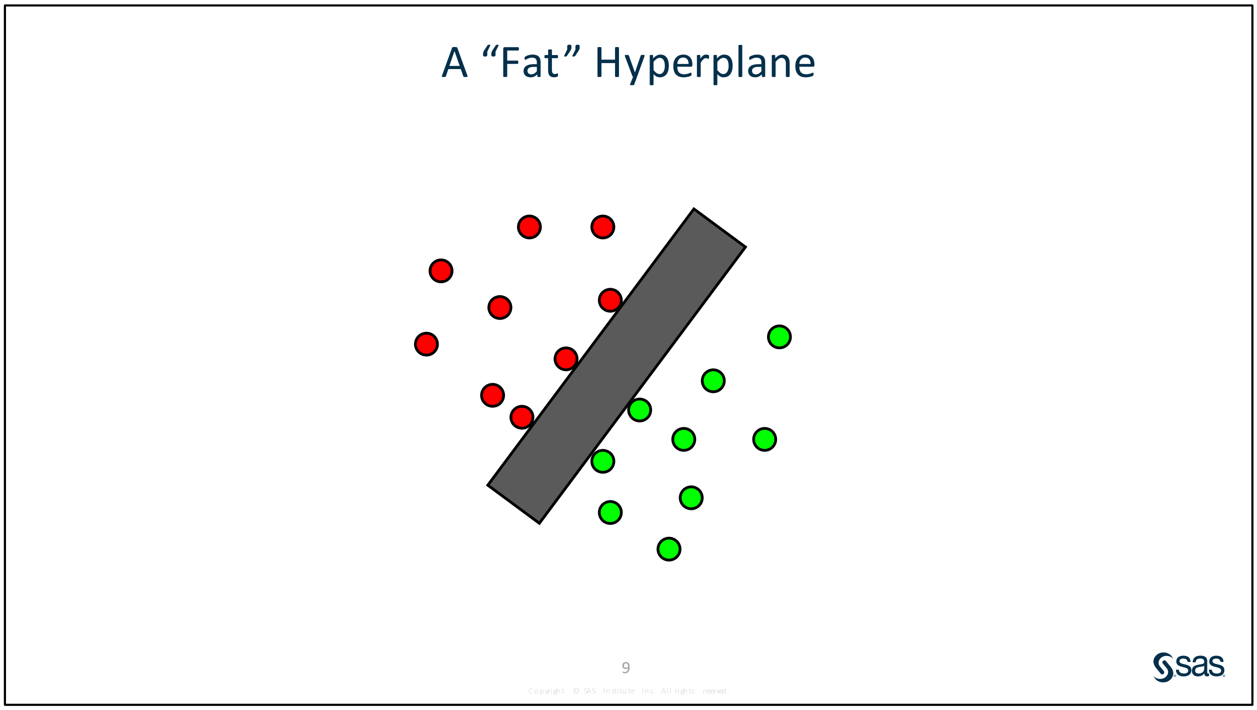
  On which side of the hyperplane (+/-) is the new data point located?

Data points located in the direction of the normal vector are diagnosed as positive. Data points on the other side of the hyperplane are diagnosed as negative.



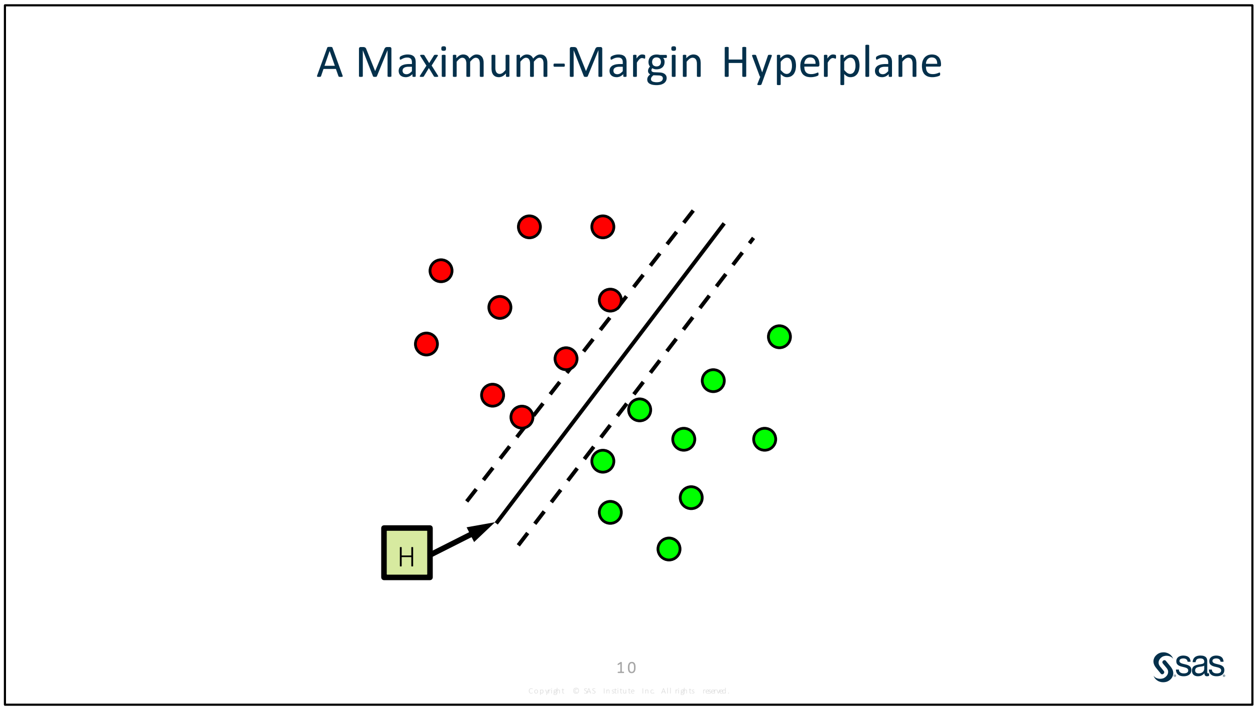
Which Hyperplane Is the Best One?

Here again is the simple illustration shown a few slides ago. If the data points are *linearly separable*, then an infinite number of separating hyperplanes (that is, classification rules) exist.



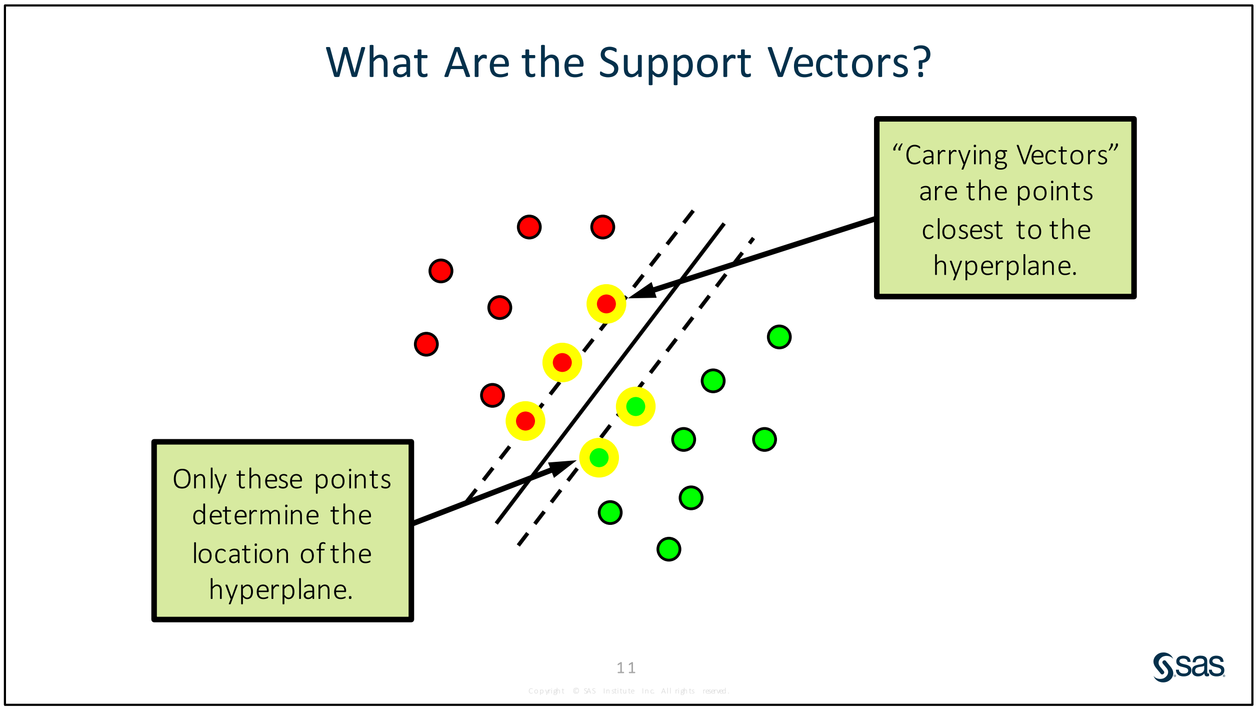
A “Fat” Hyperplane

The starting point to get to a unique solution is to think of a “fat” hyperplane. This leads to a separator that has the largest margin of error, essentially wiggle room, on either side.



A Maximum-Margin Hyperplane

Among all these hyperplanes, only one of them has the maximum margin. It is essentially the median of the fat hyperplane.



What Are the Support Vectors?

"Carrying Vectors" are the points closest to the hyperplane.

Only these points determine the location of the hyperplane.

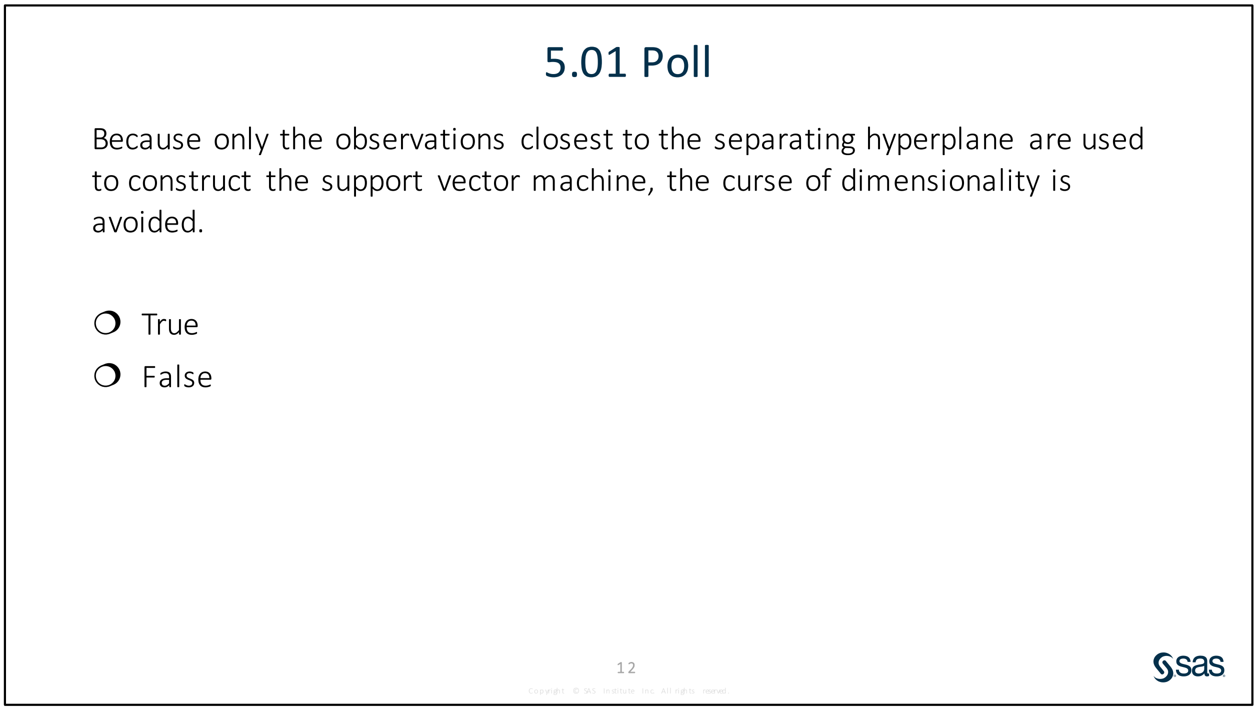
11

The normal vector for the maximum-margin separating hyperplane is given here:

$$w = \sum_{i=1}^{\#sv} \alpha_i y_i x_i^{sv}$$

(The mathematical details of this solution are provided in the next section.)

The properties of the maximum-margin hyperplane are described by the support vectors. The construction of the maximum-margin hyperplane is not explicitly dependent on the dimension of the input space. Because of this, the curse of dimensionality is avoided. The curse of dimensionality states that the more input variables a model uses, the more data points are needed to fit the model. In the illustration above, only the five points that are the carrying vectors are used to determine *w*.



5.01 Poll

Because only the observations closest to the separating hyperplane are used to construct the support vector machine, the curse of dimensionality is avoided.

○ True

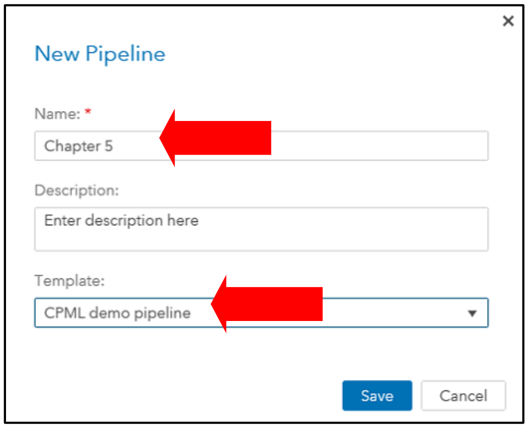
○ False

12

Building a Support Vector Machine Based on Default Settings

In this demonstration, you create a new pipeline using the CPML demo pipeline and add a Support Vector Machine node to it. You build the Support Vector Machine model using the default settings of the node.

1. Click the plus sign (**+**) next to the Chapter 4 pipeline tab to add a new pipeline.
2. In the New Pipeline window, enter **Chapter 5** in the **Name** field. For **Template**, select **CPML demo pipeline**.



New Pipeline

Name: *

Chapter 5

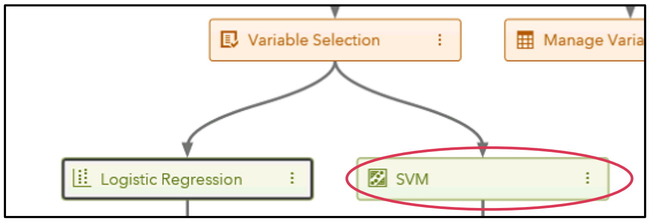
Description:

Enter description here

Template:

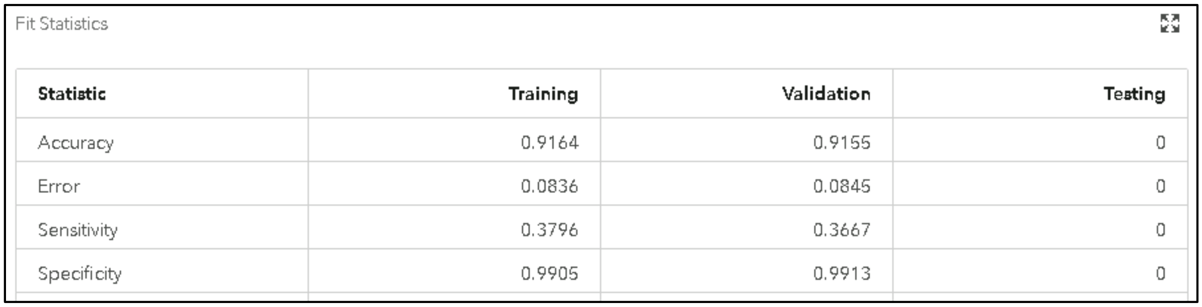
CPML demo pipeline

3. Click **Save**.
4. In the Chapter 5 pipeline, right-click the **Variable Selection** node and select **Add below** ⇨ **Supervised Learning** ⇨ **SVM**.



5. Keep all properties for the support vector machine at their defaults.
6. Run the SVM node.
7. Open the results for the Support Vector Machine model.

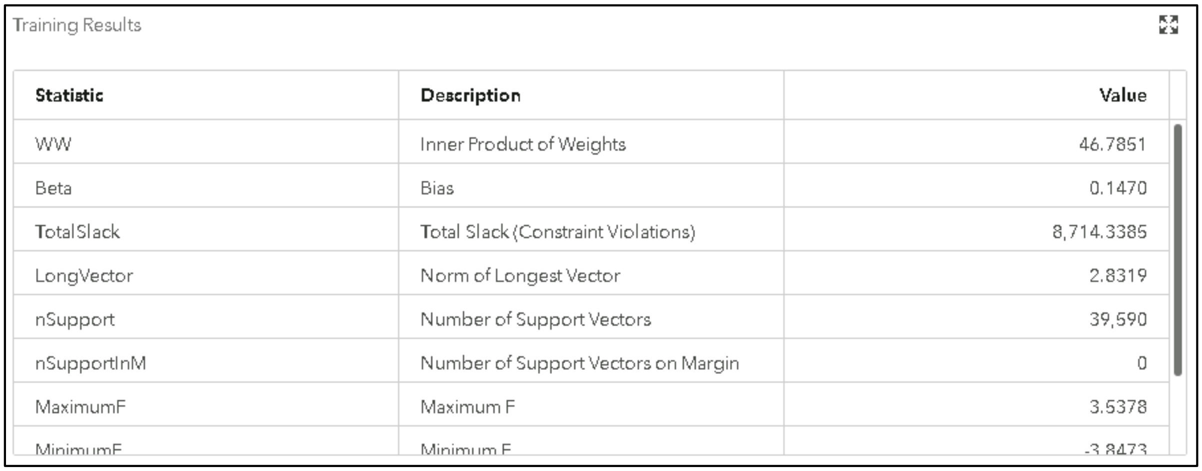
   There are several charts and plots to help you evaluate the model's performance. The first table is Fit Statistics, which presents the support vector machine's performance considering several assessment measures.



Fit Statistics

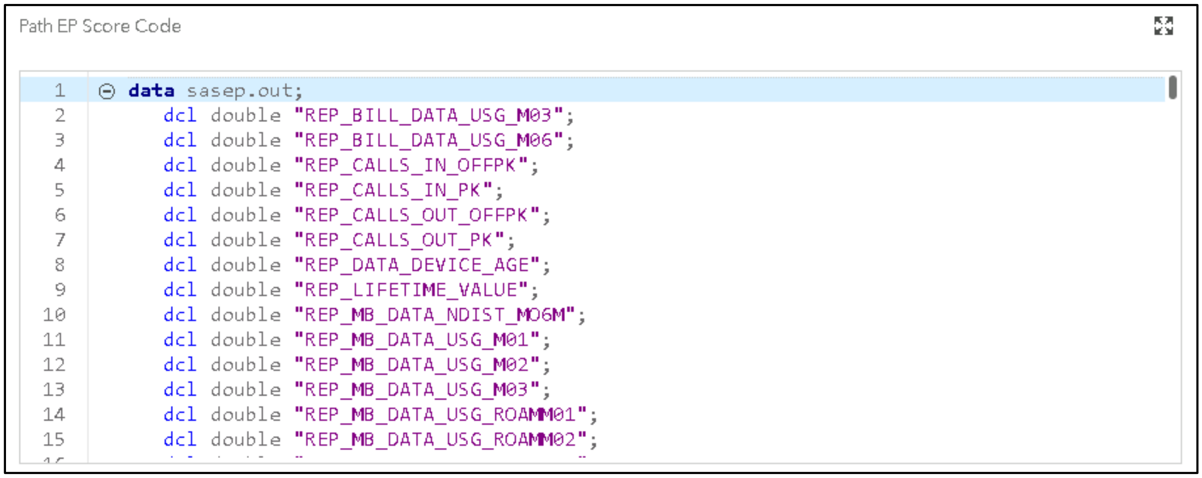
Statistic	Training	Validation	Testing
Accuracy	0.9164	0.9155	0
Error	0.0836	0.0845	0
Sensitivity	0.3796	0.3667	0
Specificity	0.9905	0.9913	0

The Training Results table shows the parameters for the final Support Vector Machine model, such as the support vectors and the margin, among others.


Training Results

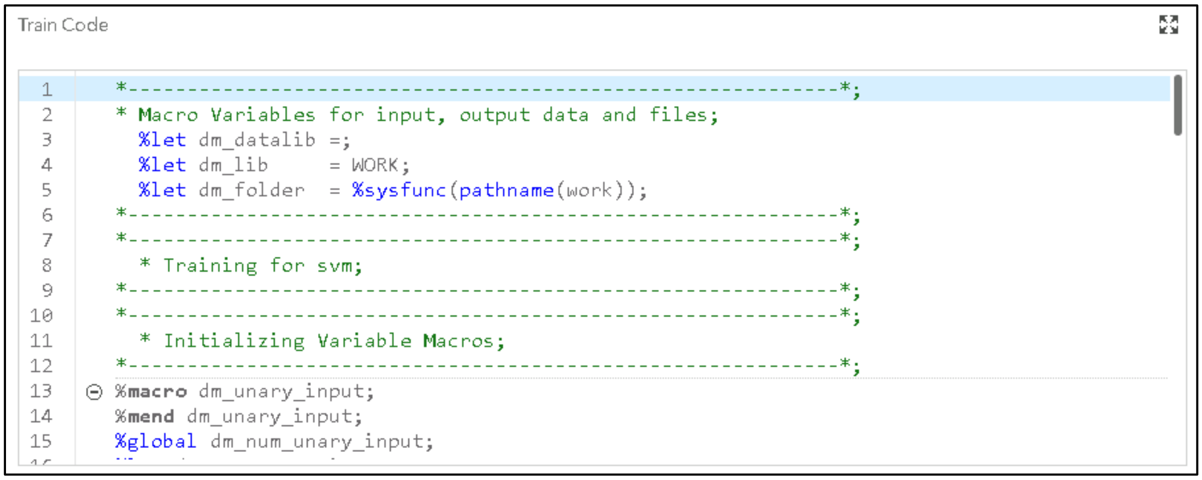
Statistic	Description	Value
WW	Inner Product of Weights	46.7851
Beta	Bias	0.1470
TotalSlack	Total Slack (Constraint Violations)	8,714.3385
LongVector	Norm of Longest Vector	2.8319
nSupport	Number of Support Vectors	39,590
nSupportInM	Number of Support Vectors on Margin	0
MaximumF	Maximum F	3.5378
MinimumF	Minimum F	-3.8473

The Path EP Score Code window shows the final score code that can be deployed in production.


Path EP Score Code

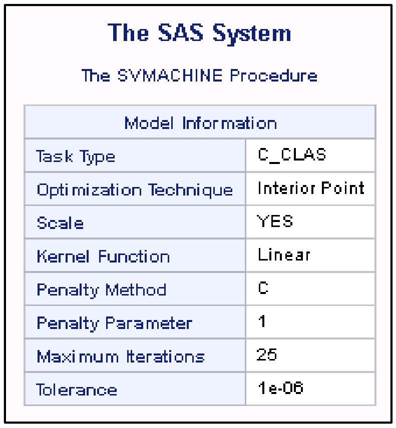
```
data sasep.out;
    dcl double "REP_BILL_DATA_USG_M03";
    dcl double "REP_BILL_DATA_USG_M06";
    dcl double "REP_CALLS_IN_OFFPK";
    dcl double "REP_CALLS_IN_PK";
    dcl double "REP_CALLS_OUT_OFFPK";
    dcl double "REP_CALLS_OUT_PK";
    dcl double "REP_DATA_DEVICE_AGE";
    dcl double "REP_LIFETIME_VALUE";
    dcl double "REP_MB_DATA_NDIST_MO6M";
    dcl double "REP_MB_DATA_USG_M01";
    dcl double "REP_MB_DATA_USG_M02";
    dcl double "REP_MB_DATA_USG_M03";
    dcl double "REP_MB_DATA_USG_ROAMM01";
    dcl double "REP_MB_DATA_USG_ROAMM02";
```

Similarly, the Train Code window shows the train code that can be used to train the model based on different data sets or in different platforms.


Train Code

```
*------------------------------------------------------------*;
* Macro Variables for input, output data and files;
  %let dm_datalib =;
  %let dm_lib     = WORK;
  %let dm_folder  = %sysfunc(pathname(work));
*------------------------------------------------------------*;
*------------------------------------------------------------*;
  * Training for svm;
*------------------------------------------------------------*;
*------------------------------------------------------------*;
  * Initializing Variable Macros;
*------------------------------------------------------------*;
%macro dm_unary_input;
%mend dm_unary_input;
%global dm_num_unary_input;
```

Finally, the Output window shows the final Support Vector Machine model parameters, the training results, the iteration history, the misclassification matrix, the fit statistics, and the predicted probability variables.

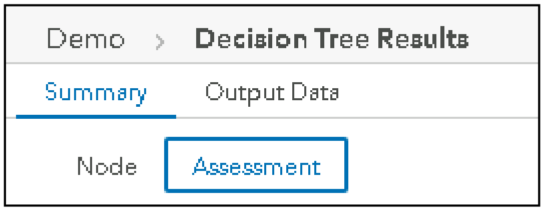


The SAS System

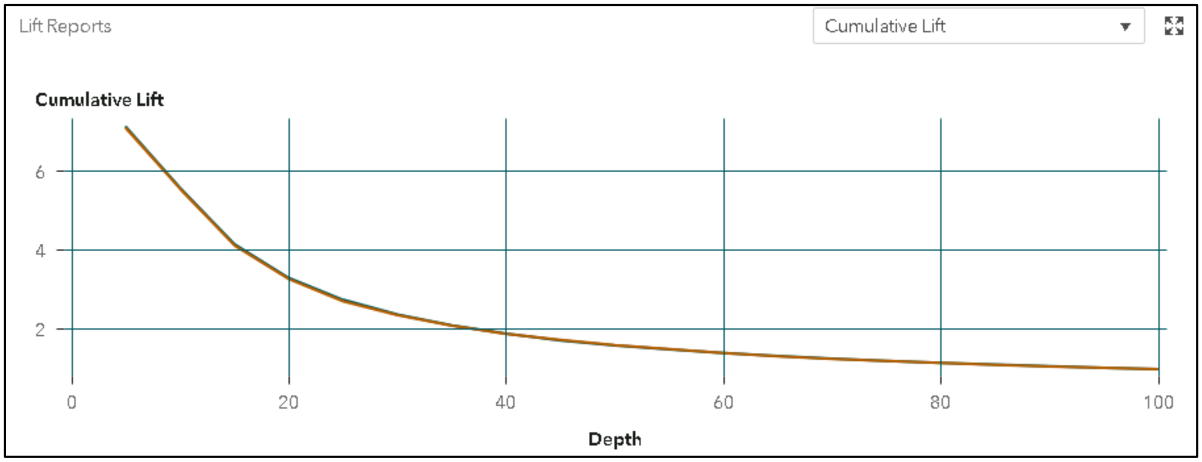
The SVMACHINE Procedure

Model Information	
Task Type	C_CLAS
Optimization Technique	Interior Point
Scale	YES
Kernel Function	Linear
Penalty Method	C
Penalty Parameter	1
Maximum Iterations	25
Tolerance	1e-06

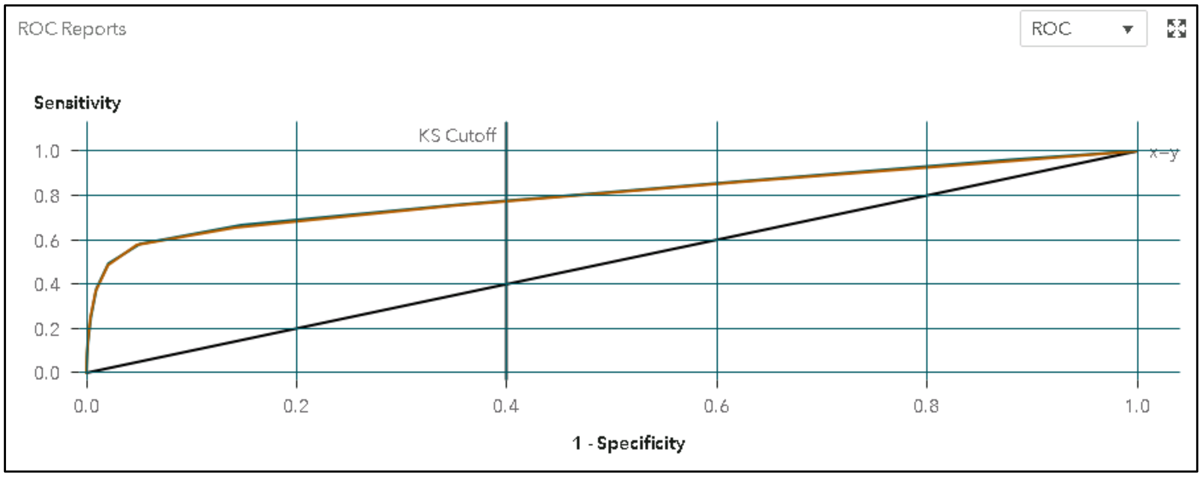
8. Click the **Assessment** tab.



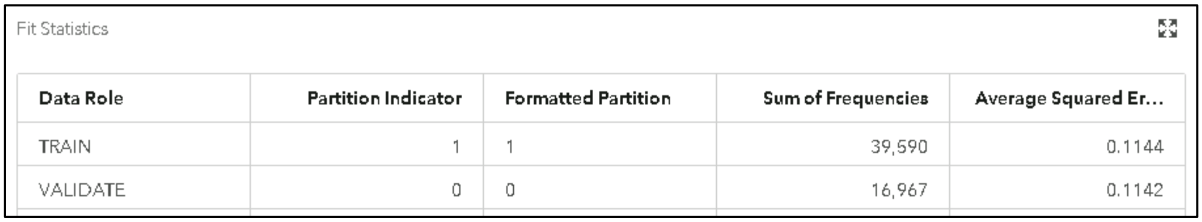
The first chart is the Cumulative Lift, showing the model's performance ordered by the percentage of the population. This chart is very useful for selecting the model based on a particular target of the customer base. It shows how much better the model is than the random events.



For a binary target, you also have the ROC Reports output, which shows the model's performance in terms of ROC curve by considering the true positive rate and the false positive rate. It is good to foresee the performance on a specific business event when all positive cases are selected. It shows that the model's performance based on the positive cases were predicted right and the positive cases were predicted wrong. ROC is very useful for deployment.



Finally, you have the Fit Statistics output, which shows the model's performance based on some assessment measures, such as average square error.

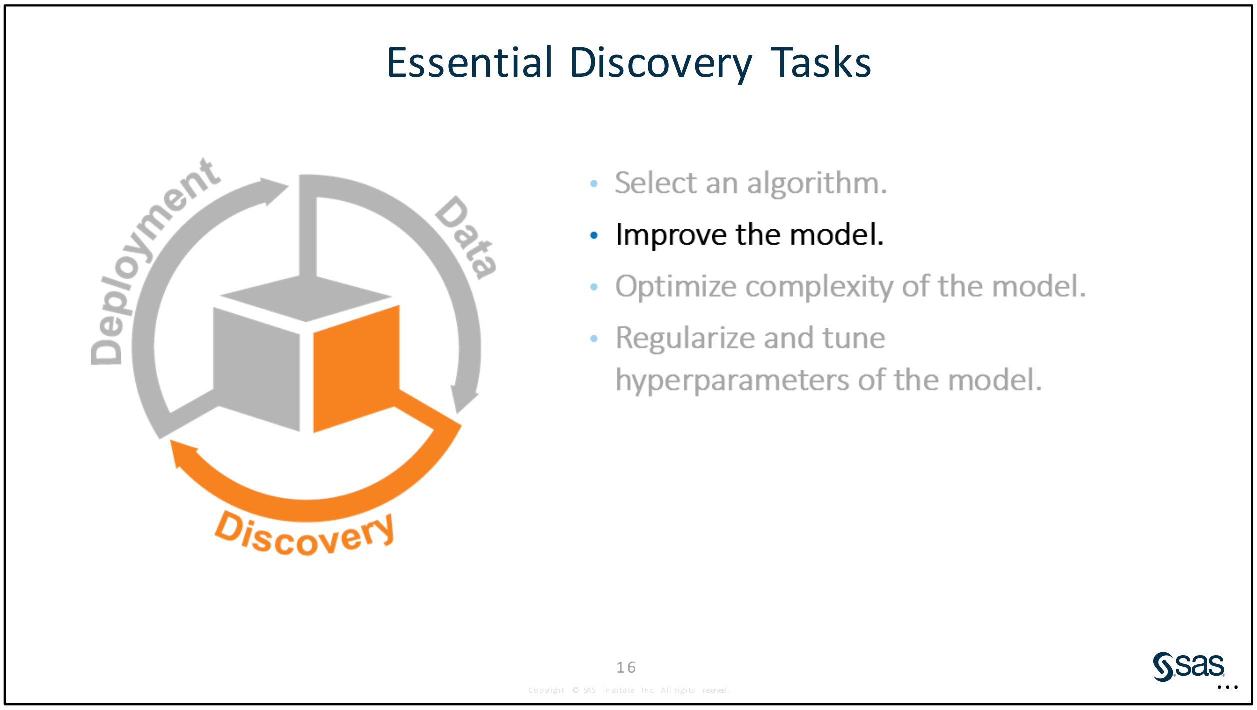


Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.1144
VALIDATE	0	0	16,967	0.1142

9. The Fit Statistics table shows an average square error of 0.1142 on the VALIDATE partition.
10. Close the Results window.

End of Demonstration

5.2 Methods of Solution

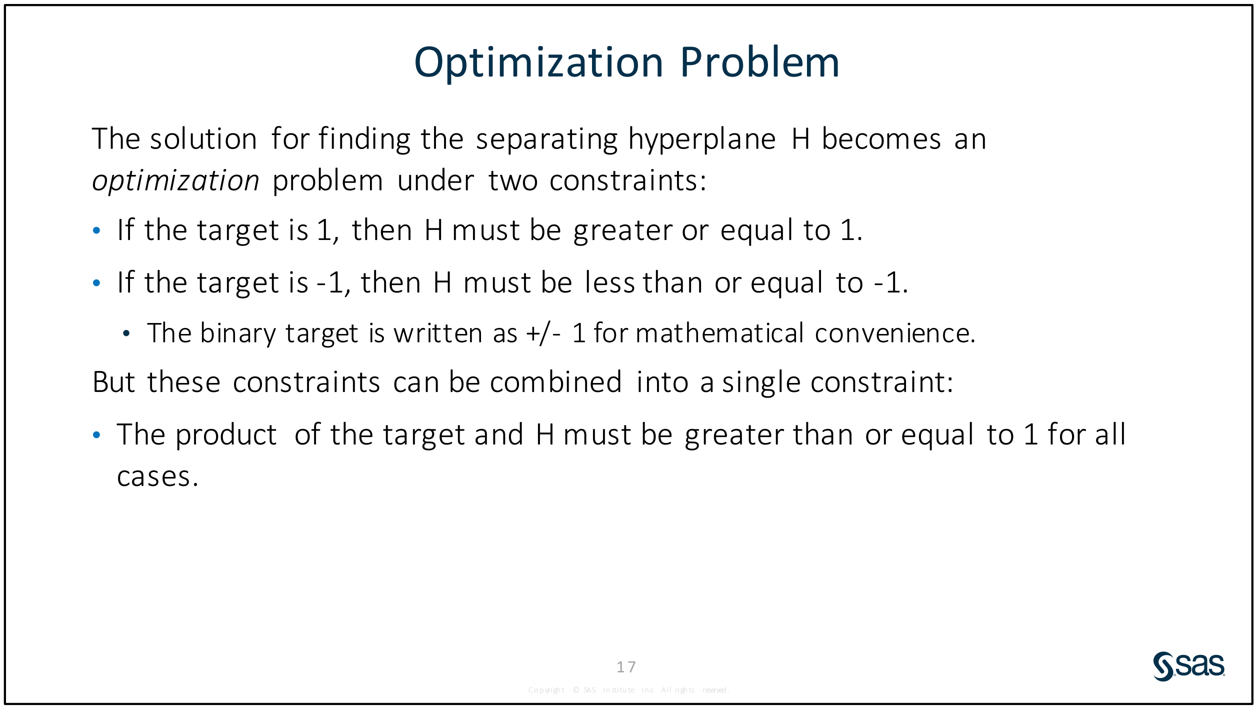


Essential Discovery Tasks

- Select an algorithm.
- Improve the model.
- Optimize complexity of the model.
- Regularize and tune hyperparameters of the model.

16

One of the methods to improve support vector machine models is by changing the kernel functions and the penalty.



Optimization Problem

The solution for finding the separating hyperplane H becomes an *optimization* problem under two constraints:

- If the target is 1, then H must be greater or equal to 1.
- If the target is -1, then H must be less than or equal to -1.
  - The binary target is written as +/- 1 for mathematical convenience.

But these constraints can be combined into a single constraint:

- The product of the target and H must be greater than or equal to 1 for all cases.

17

If the target variable equals 1, then *H* must be greater than or equal to 1. If the target is -1, then *H* must be less than or equal to -1. The optimal hyperplane satisfies these conditions and also has minimal norm. The denoting of the binary target as +1 or -1 is simply for ease in mathematical details. This trick enables the combination of the two constraints into a single constraint. Optimization problems with a single constraint are mathematically easier to solve than optimization problems under two constraints.

Details:

Mathematically, these constraints are written as follows:

$$\langle w, x_i \rangle + b \geq 1$$

if y_i = 1 and

$$\langle w, x_i \rangle + b \leq -1$$

if y_i = -1.

However, these two constraints can be combined into a single constraint where

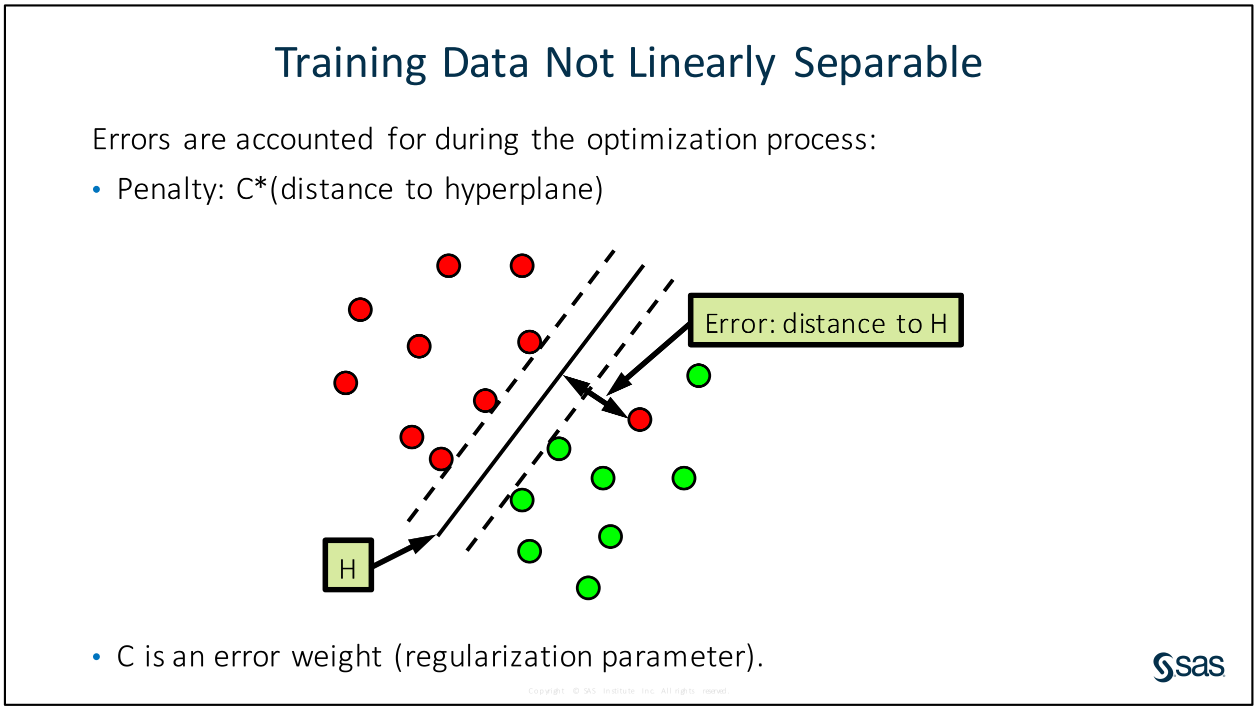
$$y_i \cdot \left(\langle w, x_i \rangle + b \right) \geq 1$$

for *i* = 1, 2, …*n*.

The width of this *maximum margin hyperplane* is determined by the usual calculation of a point to a line. In general, the distance from a point (x_0,y_0) to a line $Ax + By + C = 0$, is given by $|Ax_0 + By_0 + C|/sqrt(A^2 + B^2)$.

Using this calculation, the maximum margin hyperplane is found by maximizing 2/||*w*||, where ||*w*|| is the *norm* of the vector *w*, which is defined as ||w||= *sqrt*(*w'w*). The norm of a vector is a measure of length.

Maximizing 2/||*w*|| is equivalent to minimizing ||*w*||. Because ||*w*|| is defined by using a square root, it becomes mathematically simpler to minimize the square of ||*w*||; the solution is the same.



Training Data Not Linearly Separable

Errors are accounted for during the optimization process:

- Penalty: C*(distance to hyperplane)

- C is an error weight (regularization parameter).

If the data points are not linearly separable, we have a so-called *soft margin* hyperplane. In this case, we need to account for errors that the separating hyperplane might make. During the optimization process, the distance between a point in error and the hyperplane is typically denoted by *ξ*.

Details: The Solution

Given the need to account for errors, the optimization problem is solved by minimizing:

$$\| w \|^2 + C \cdot \sum_i \xi_i$$

under the single constraint

$$y_i \cdot \left(\langle w, x_i \rangle + b \right) \geq 1 - \xi_i, \quad \xi_i \geq 0 .$$

The method used to solve the optimization problem is the Lagrange approach. Here, Lagrange multipliers $\alpha_i \geq 0$ are introduced. They summarize the problem in a Lagrange function. Constraints: $\xi_i \geq 0$ and $\alpha_i \geq 0$, so you must find the saddle point of the Lagrange function.

For the optimization problem above, the Lagrange function becomes

$$L(w, b, \alpha, \xi) = \frac{1}{2} \| w \|^2 + C \cdot \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i \left(\xi_i + y_i \left(\langle w, x_i \rangle + b \right) - 1 \right) .$$

In order to find the saddle point, $L(w, b, \alpha, \xi)$ is minimized with respect to *w*, *b*, and *ξ* but maximized with respect to α_i.

Self-Study: The Lagrange Approach

Take the following derivatives of the Lagrange function:

$$\frac{\partial}{\partial b} L(w,b,\alpha,\xi) = 0 \quad , \quad \frac{\partial}{\partial w} L(w,b,\alpha,\xi) = 0$$

and obtain

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \quad , \quad w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

This leads to the so-called ***dual problem***. Maximize

$$W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle$$

under the constraints

$$0 \le \alpha_i \le C \text{ and } \sum_{i=1}^{n} \alpha_i y_i = 0$$

After plugging back into the Lagrange function and reformulating, you have the classification function:

$$f(x_{new}) = sign\left(\langle w, x_{new} \rangle + b\right)$$

$$= sign\left(\sum_{i=1}^{n} \alpha_i y_i \langle x_i, x_{new} \rangle + b\right)$$

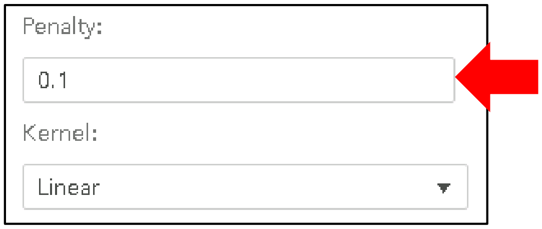
Changing the Methods of Solution for a Support Vector Machine

In this demonstration, you change the default settings of the Support Vector Machine node in the Chapter 5 pipeline. You modify the methods of solution parameters for the Support Vector Machine node.

1. Recall that the average square error of the previous model, based on the default settings, was 0.1142 on the VALIDATE partition.

   Try to improve the support vector machine performance by changing some of the default settings assigned to the methods of solution.

2. Change the **Penalty** property from **1** to **0.1**.



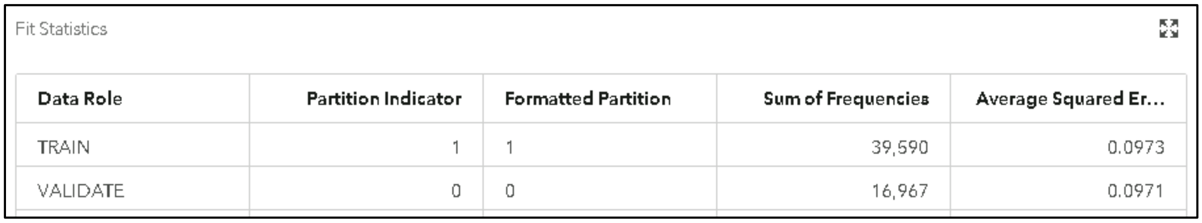
Penalty:

0.1

Kernel:

Linear

3. Run the Support Vector Machine node.
4. Open the results for the Support Vector Machine node.
5. Click the **Assessment** tab. See the Fit Statistics window.



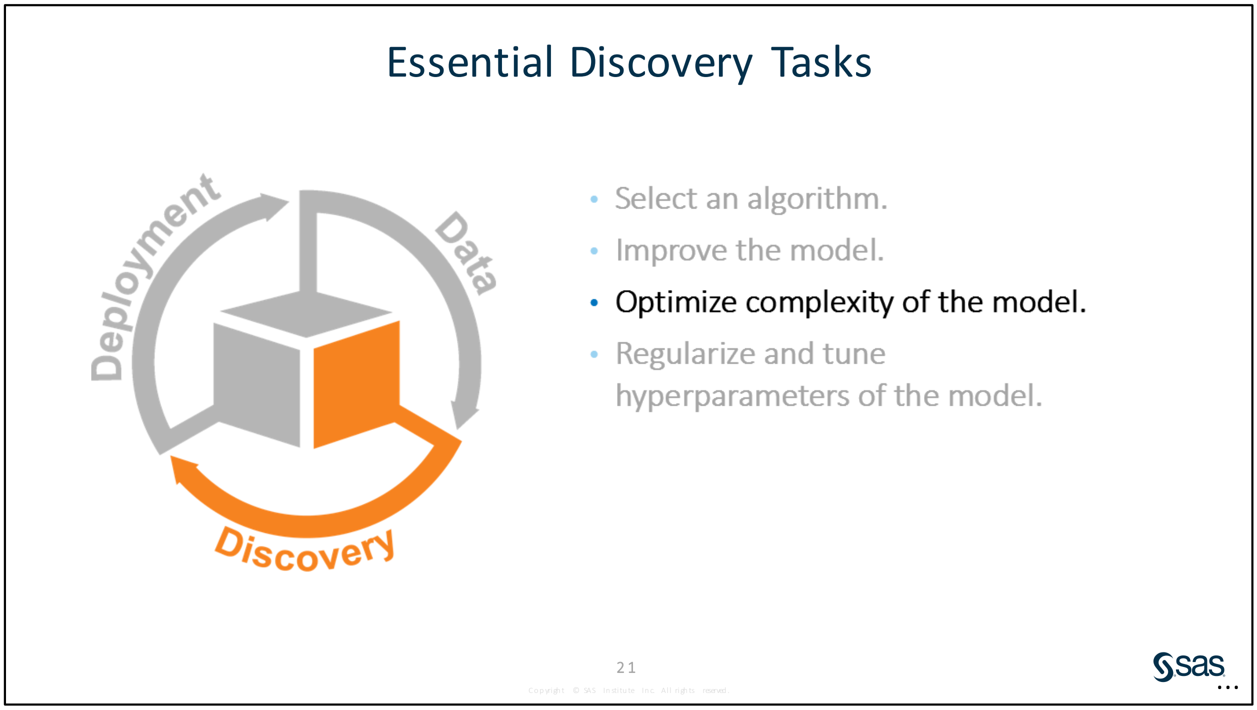
Fit Statistics

Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.0973
VALIDATE	0	0	16,967	0.0971

6. The average square error for the tuned Support Vector Machine model is 0.0971 on the VALIDATE partition. This fit statistic is much better (approximately 15%) than the first model, which used the default settings.
7. Close the Results window.

End of Demonstration

5.3 Nonlinear Classifier: Kernel Trick

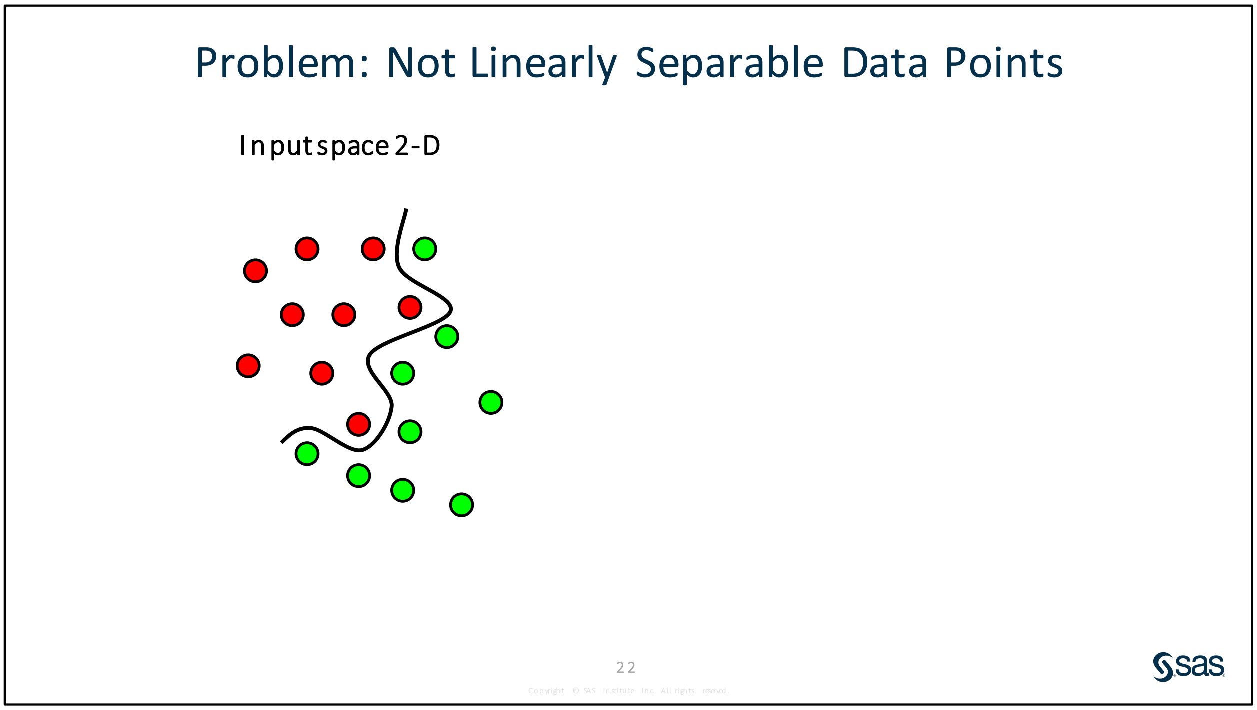


Essential Discovery Tasks

- Select an algorithm.
- Improve the model.
- Optimize complexity of the model.
- Regularize and tune hyperparameters of the model.

21

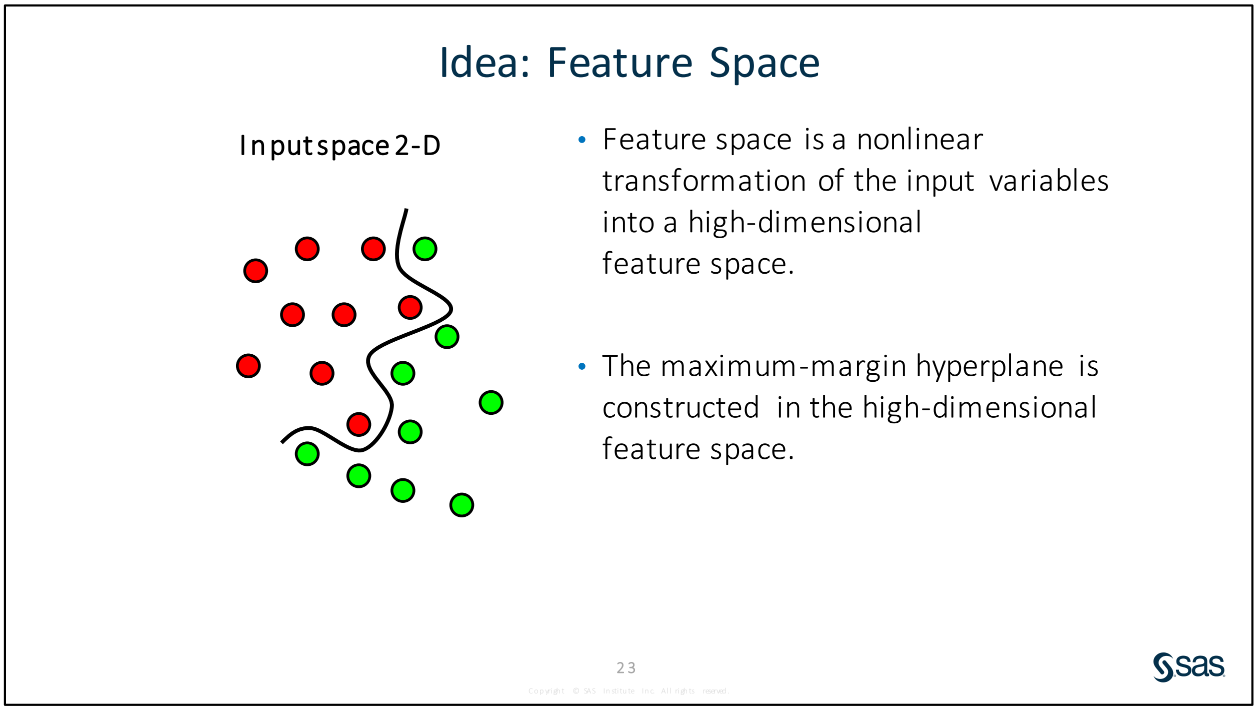
A method to optimize the complexity in Support Vector Machine models is by changing the margins.



Problem: Not Linearly Separable Data Points

22

In most realistic scenarios, not only is data not linearly separable, but a soft margin classifier would make too many mistakes to be a viable solution.

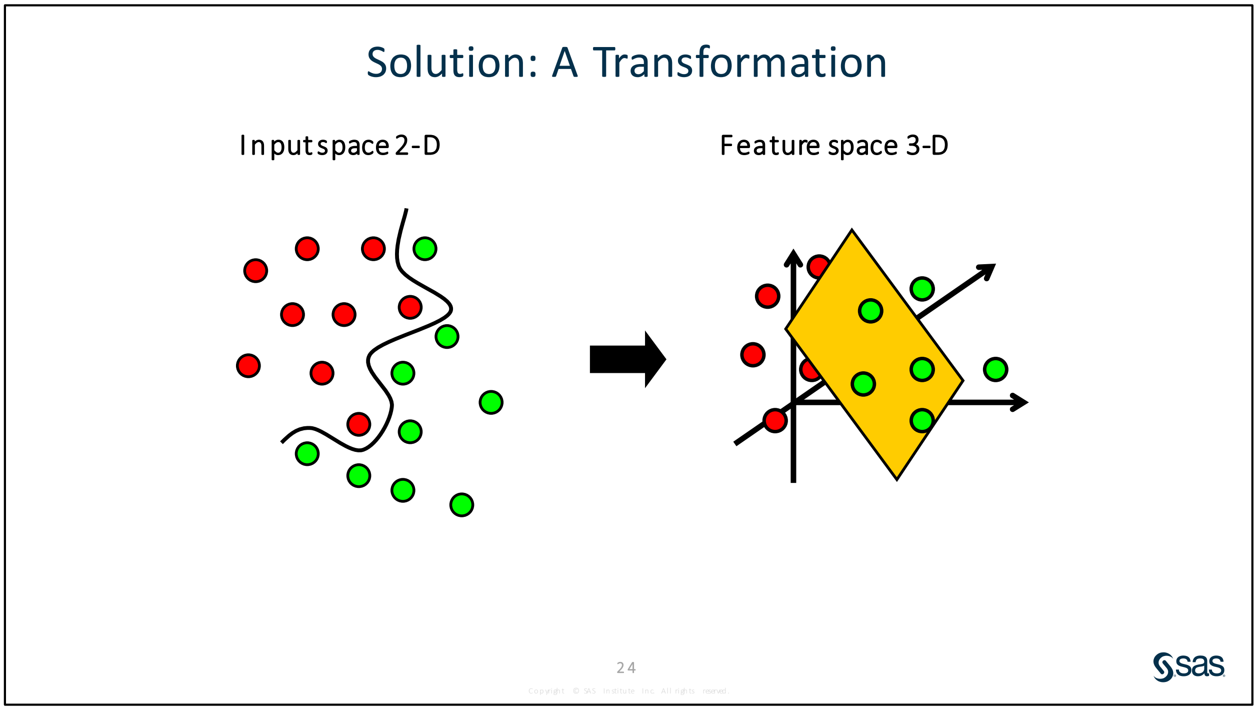


Idea: Feature Space

- Feature space is a nonlinear transformation of the input variables into a high-dimensional feature space.
- The maximum-margin hyperplane is constructed in the high-dimensional feature space.

23

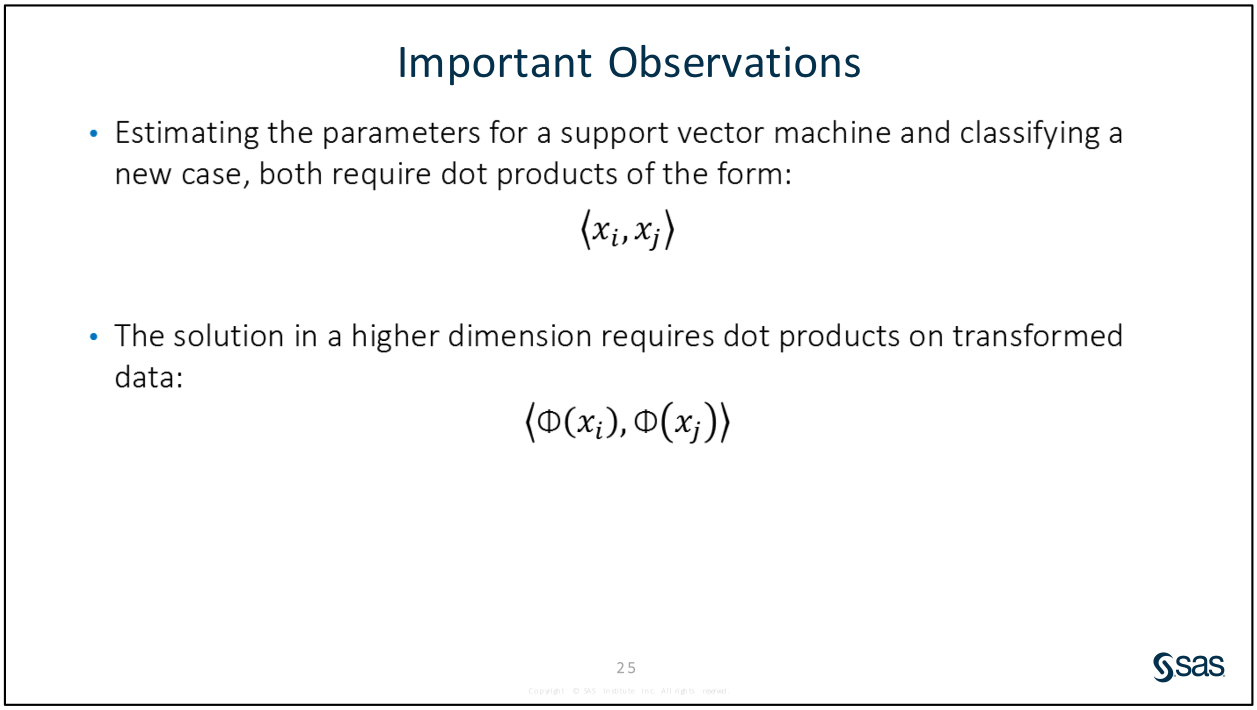
A solution is most real-world cases requires transforming the data to a higher dimension and then finding the maximum margin hyperplane in this higher dimension.



Solution: A Transformation

24

Here is an example that is not linearly separable in two dimensions, but it is easy to separate in three dimensions. This can be generalized to higher dimensions.



Important Observations

- Estimating the parameters for a support vector machine and classifying a new case, both require dot products of the form:

$$\langle x_i, x_j \rangle$$

- The solution in a higher dimension requires dot products on transformed data:

$$\langle \Phi(x_i), \Phi(x_j) \rangle$$

25

The original data points occur only in dot products. So, whether solving for the parameters of *H* or when scoring a new observation, the calculations depend on dot products.

Details:

Here is the equation used to solve the Dual Optimization problem:

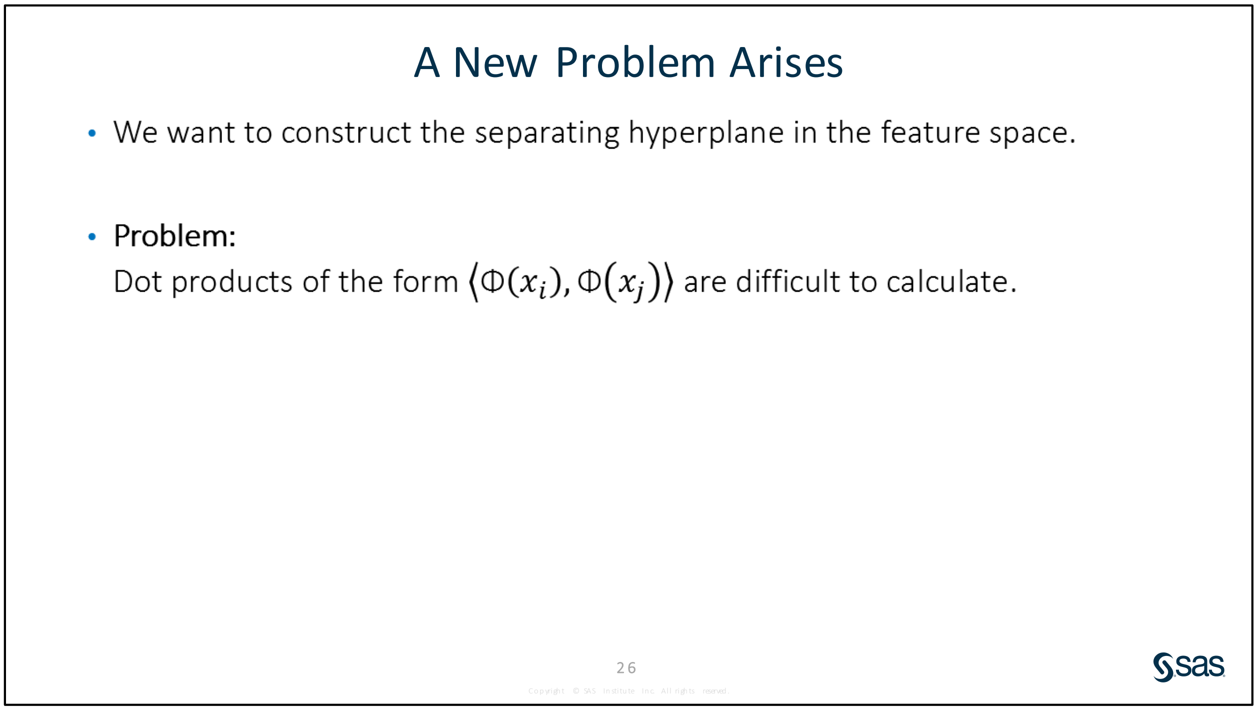
$$W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \left\langle x_i, x_j \right\rangle .$$

Here is the equation used to classify a new case:

$$f(x_{new}) = sign \left(\sum_{i=1}^{n} \alpha_i y_i \left\langle x_i, x_{new} \right\rangle + b \right) .$$

Both equations rely on dot products of the data.

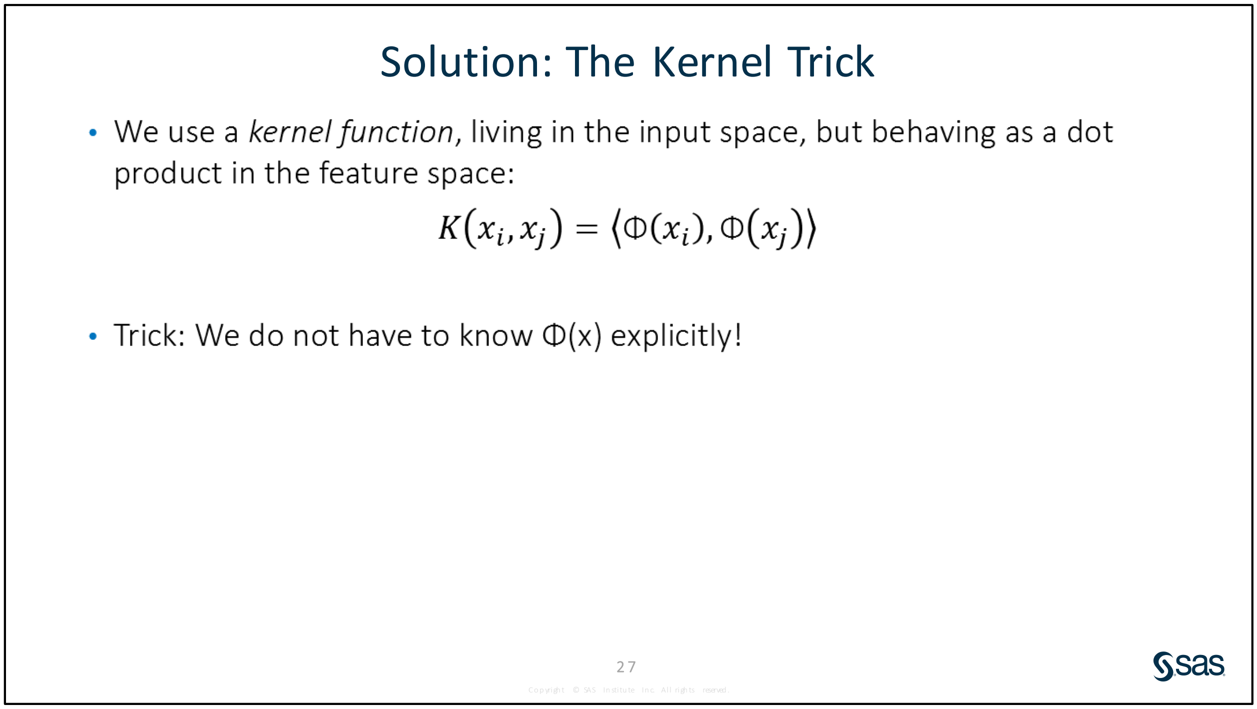
For data that are not linearly separable, the data points are transformed to a *feature space* with a function Φ. Then we separate the data points $\Phi(x)$ in the feature space. Usually, the dimension of the feature space is ***much*** higher than the dimension of the input space.



A New Problem Arises

- We want to construct the separating hyperplane in the feature space.
- Problem:
  Dot products of the form $\langle \Phi(x_i), \Phi(x_j) \rangle$ are difficult to calculate.

If the classification really is easier in the high-dimensional feature space, you want to construct the separating hyperplane there. This requires dot product calculations in the feature space. This creates a problem, as dot products in feature spaces are mathematically difficult to calculate.



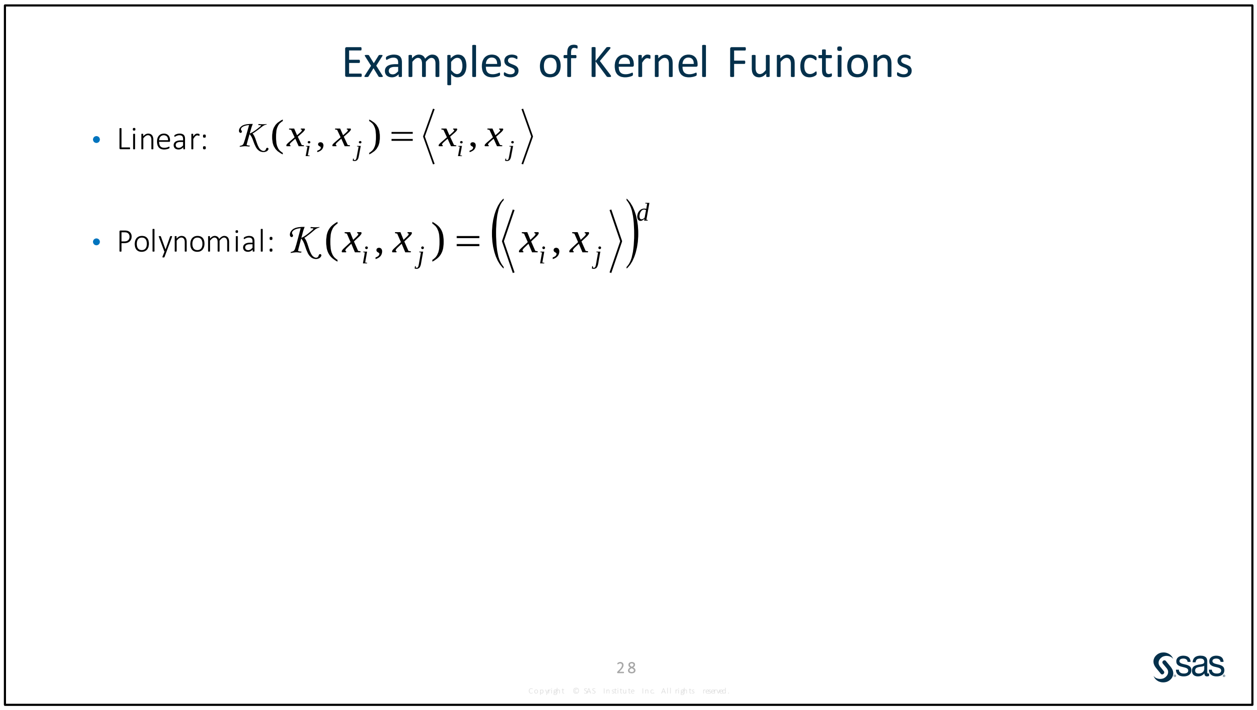
Solution: The Kernel Trick

- We use a *kernel function*, living in the input space, but behaving as a dot product in the feature space:

$$K(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle$$

- Trick: We do not have to know Φ(x) explicitly!

You can overcome the curse of dimensionality because you need to calculate only inner products between vectors in the feature space. $\Phi(x)$ is the transformation from the input space to the feature space. You do not need to perform the mapping explicitly. If K satisfies Mercer's theorem, it describes an inner product.

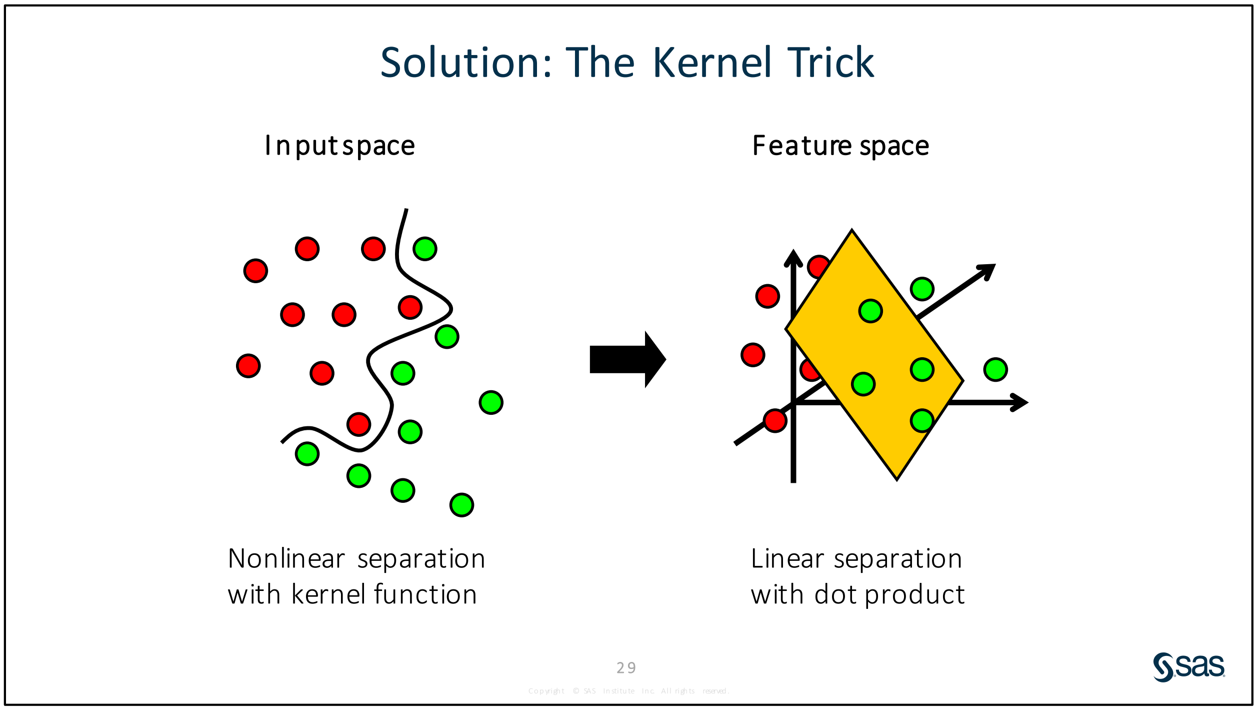


Examples of Kernel Functions

- Linear: $\mathcal{K}(x_i, x_j) = \langle x_i, x_j \rangle$
- Polynomial: $\mathcal{K}(x_i, x_j) = \left(\langle x_i, x_j \rangle\right)^d$

28

These are the two kernel functions available in Model Studio. Why do we call it a trick? You do not have to know exactly what the feature space looks like. It is enough to specify the kernel function as a measure of similarity. You do not perform the exact kernel calculations but consider the result. Still, you have the geometric interpretation in the form of a separating hyperplane (that is, more transparency as for a neural network).



Solution: The Kernel Trick

Input space

Feature space

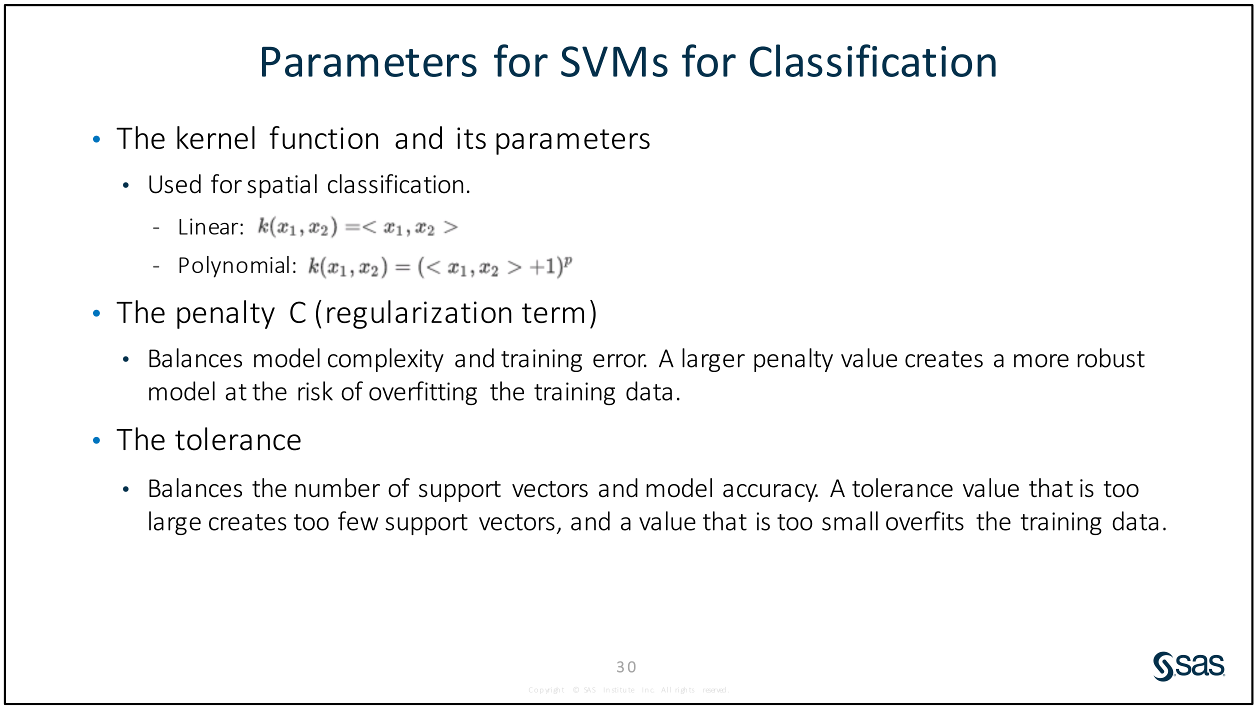
Nonlinear separation with kernel function

Linear separation with dot product

29

In the slide above, the points are not linearly separable in two dimensions but are in three dimensions. Using a kernel function in the input space when there is nonlinear separation is equivalent to performing dot products in a higher-dimensional feature space that does have linear separation.

Support vector machines can model nonlinear data. Compared to neural networks, at times support vector machines are faster and might not stick to local minima.



Parameters for SVMs for Classification

- The kernel function and its parameters
  - Used for spatial classification.
    - Linear: $k(x_1, x_2) = < x_1, x_2 >$
    - Polynomial: $k(x_1, x_2) = (< x_1, x_2 > +1)^p$
- The penalty C (regularization term)
  - Balances model complexity and training error. A larger penalty value creates a more robust model at the risk of overfitting the training data.
- The tolerance
  - Balances the number of support vectors and model accuracy. A tolerance value that is too large creates too few support vectors, and a value that is too small overfits the training data.

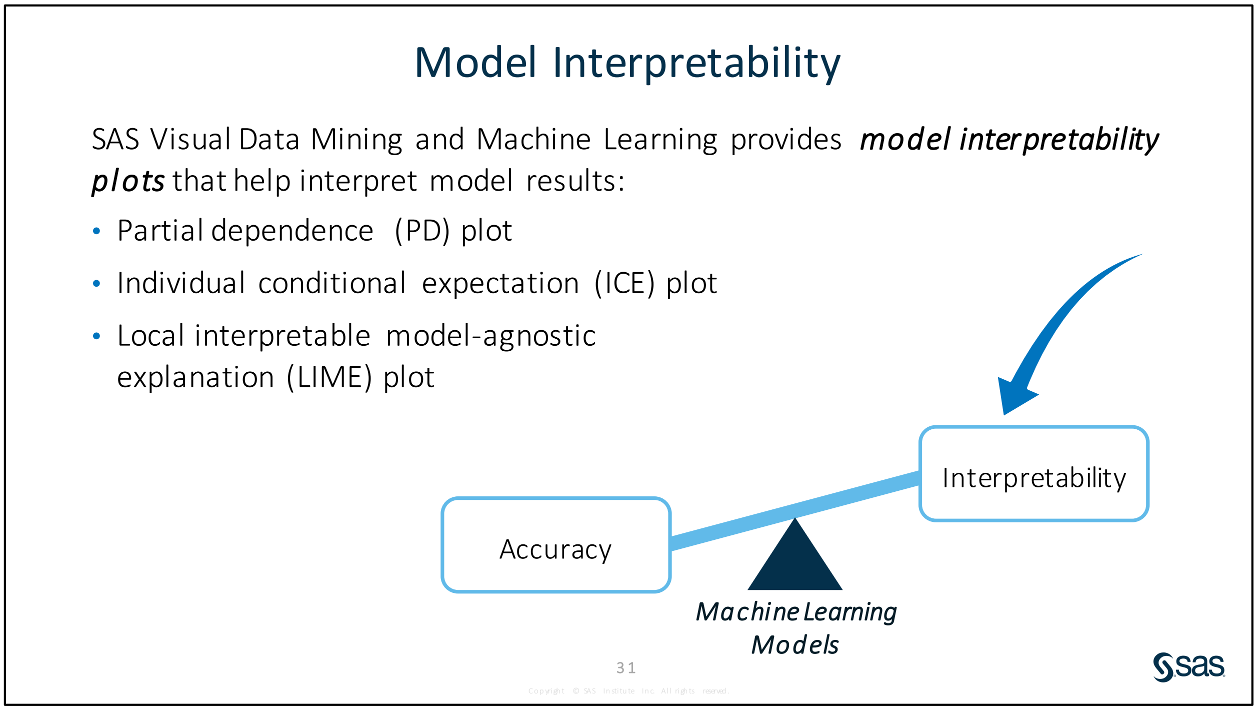
30

In Model Studio, for polynomial kernels, only degrees of 2 and 3 are available. The kernel function is used for spatial classification.

- Linear K(u, v) = $u^T v$.
- Quadratic K(u, v) = $(u^T v + 1)^2$. The 1 is added to avoid zero-value entries in the Hessian matrix.
- Cubic K(u, v) = $(u^T v + 1)^3$. The 1 is added to avoid zero-value entries in the Hessian matrix.

The penalty value balances model complexity and training error. A larger penalty value creates a more robust model at the risk of overfitting the training data.

The tolerance value balances the number of support vectors and model accuracy. A tolerance value that is too large creates too few support vectors, and a value that is too small overfits the training data. It is a user-defined value to control the absolute error of the object function. The iteration stops if the absolute error is less than or equal to the tolerance value.



Model Interpretability

SAS Visual Data Mining and Machine Learning provides ***model interpretability plots*** that help interpret model results:

- Partial dependence (PD) plot
- Individual conditional expectation (ICE) plot
- Local interpretable model-agnostic explanation (LIME) plot

31

There is almost always a trade-off in terms of accuracy versus interpretability. Machine learning algorithms are good at generating very accurate (and generalizable) predictive models using quite complex combinations of mathematical and logical elements. They provide very good predictions, but it can be nearly impossible to understand how they arrived at those predictions or in general what the behavior or trend of the model is. This is a big problem in regulated industries and other applications where it is important to be able to explain ***why*** a model gave a certain answer. So, model interpretability (understanding the predictions) is the usual criticism of machine learning models. As machine learning models become more sophisticated, the ability to quickly and accurately interpret these models can diminish.

This criticism stems from the complex parameterizations found in the model. Although it is true that little insight can be gained by analyzing the actual parameters of the model, much can be gained by analyzing the resulting prediction decisions. SAS Visual Data Mining and Machine Learning provides three plots that help users interpret model results:

Partial Dependence – A PD plot depicts the functional relationship between the model inputs and the model's predictions. A PD plot shows how the model's predictions partially depend on the values of the input variables of interest. To create a one-way PD plot, identify the plot variable and the complementary variables. Next, create a replicate of the training data for each unique value of the plot variable. In each replicate, the plot variable is replaced by the current unique value. Finally, score each replicate with your model and compute the average predicted value within each replicate. The final result is a view of how the prediction changes with respect to the plot variable.

Individual Conditional Expectation – An ICE plot presents a disaggregation of the PD plot to reveal interactions and differences at the observation level. The ICE plot is generated by choosing a plot variable and replicating each observation for every unique value of the plot variable. Then, each replicate is scored. SAS Visual Data Mining and Machine Learning creates a segmented ICE plot. A segmented ICE plot is created from a cluster of observations instead of on individual observations.

Local Interpretable Model-Agnostic Explanations – A LIME plot creates a localized linear regression model around a particular observation based on a perturbed sample set of data. That is, near the observation of interest, a sample set of data is created. This data set is based on the distribution of the original input data. The sample set is scored by the original model, and sample observations are weighted based on proximity to the observation of interest. Next, variable selection is performed using the LASSO technique. Finally, a linear regression is created to explain the relationship between the perturbed input data and the perturbed target variable. The final result is an easily interpreted linear regression model that is valid near the observation of interest.

Note: Each of these plots work for all models and are used to compare results across many different models.

Note: For more details, visit the blog series at https://blogs.sas.com/content/tag/interpretability/.

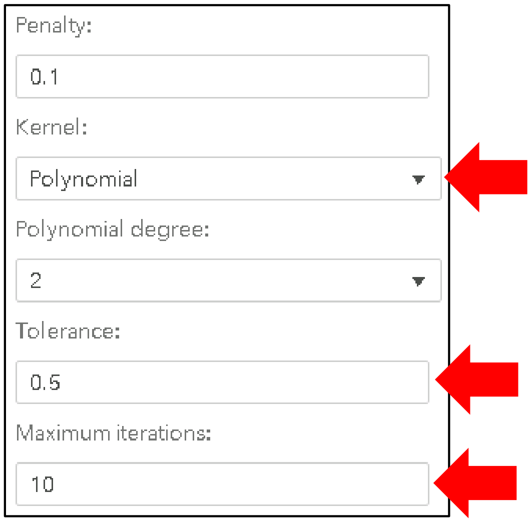
Changing the Kernel Function for a Support Vector Machine and Adding Model Interpretability

In this demonstration, you change the previous settings of the Support Vector Machine node in the Chapter 5 pipeline. You modify the kernel function and other parameters and compare this model performance to the other model in the pipeline. Later, you use the Model Interpretability capability to add some explanation to the support vector machine model.

1. To recall, the previous model, based on changings in the methods of solution, achieved an average square error of 0.0971 on the VALIDATE partition. This fit statistic was better than the first model (which used the default settings) by approximately 15%.

   Try to improve the support vector machine performance by changing some of the default settings assigned to the kernel function parameters.

2. For **Kernel**, change the function from **Linear** to **Polynomial**. Leave **Polynomial degree** as **2**.
3. For **Tolerance**, change **0.000001** to **0.5**.
4. For **Maximum iterations**, change **25** to **10**.



Penalty:

0.1

Kernel:

Polynomial

Polynomial degree:

2

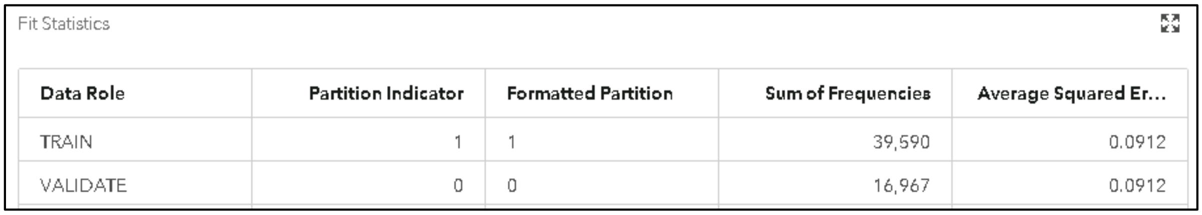
Tolerance:

0.5

Maximum iterations:

10

5. Run the Support Vector Machine node.
6. Open the results for the Support Vector Machine model.
7. Click the **Assessment** tab. See the Fit Statistics window.



Fit Statistics

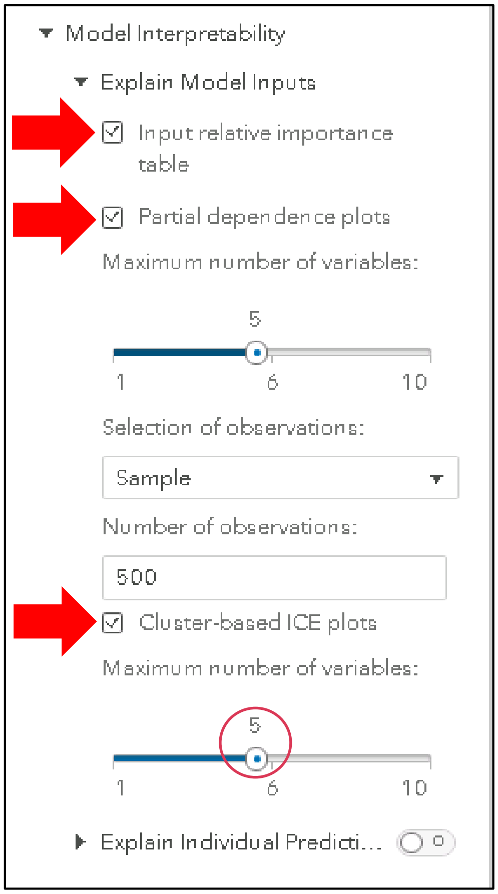
Data Role	Partition Indicator	Formatted Partition	Sum of Frequencies	Average Squared Er...
TRAIN	1	1	39,590	0.0912
VALIDATE	0	0	16,967	0.0912

The average square error for the tuned Support Vector Machine model is 0.0912 on the VALIDATE partition. This fit statistic is slightly better than the previous model, by approximately 6%.

8. Close the Results window.

   Some improvement in the model performance has been observed. However, model interpretation is still a challenging task in machine learning models that include support vector machines.

9. Under **Model Interpretability**, select all the three check boxes: **Input relative importance table**, **Partial dependence plots**, and **Cluster-based ICE plots**.



▾ Model Interpretability

▾ Explain Model Inputs

☑ Input relative importance table

☑ Partial dependence plots

Maximum number of variables: 5

Selection of observations: Sample

Number of observations: 500

☑ Cluster-based ICE plots

Maximum number of variables: 5

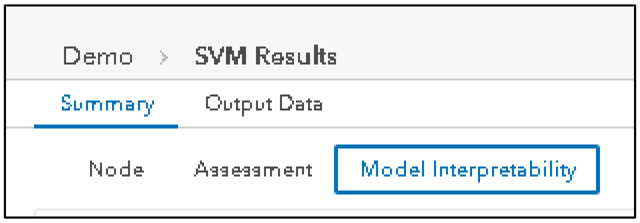
10. Under **Cluster-based ICE plots**, change **Maximum number of variables** from **3** to **5**. This ensures that the PD plots (the default is already 5) and the ICE plots are created for the five most important variables in the Input relative importance table.

11. **Explain Individual Predictions** specifies whether to use a cluster-based version of the LIME method to explain the model predictions. For brevity, leave it cleared, which is default.

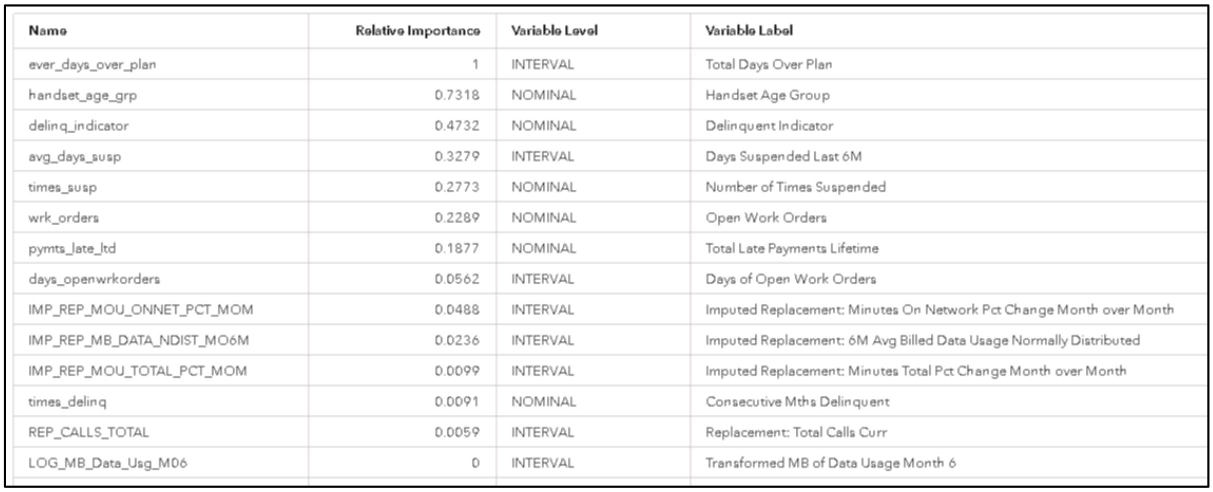
12. Run the Support Vector Machine node.

13. Open the results for the Support Vector Machine model.

14. Now you see an additional tab along with the Node and Assessment tabs. Click the **Model Interpretability** tab.



15. Click ⛶ to expand the Input Relative Importance table.

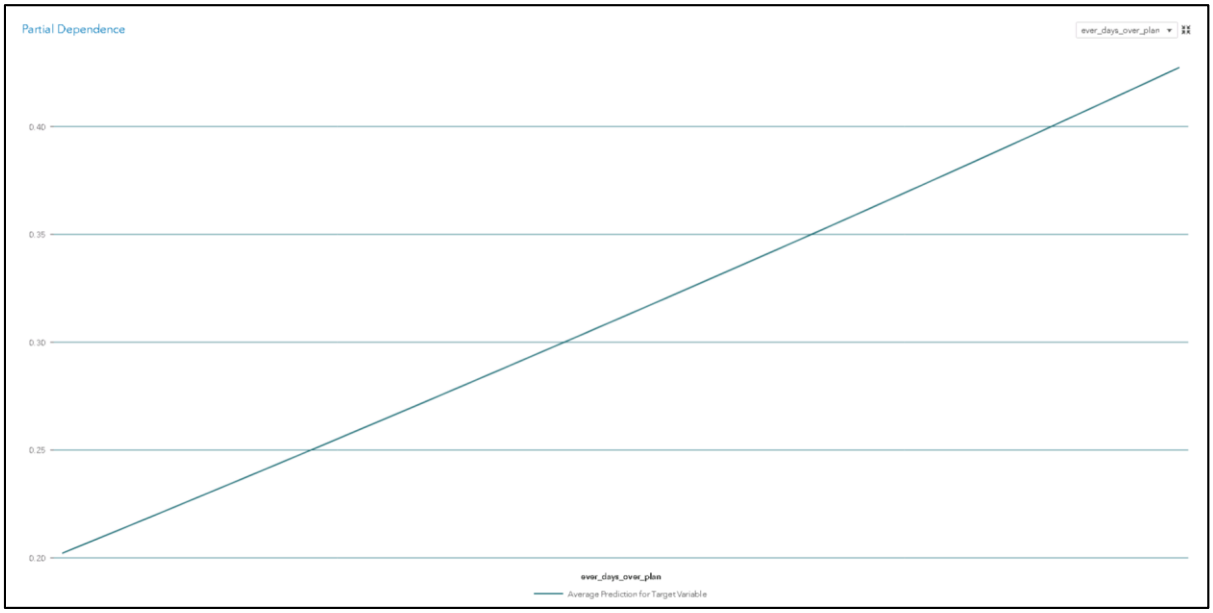


Name	Relative Importance	Variable Level	Variable Label
ever_days_over_plan	1	INTERVAL	Total Days Over Plan
handset_age_grp	0.7318	NOMINAL	Handset Age Group
delinq_indicator	0.4732	NOMINAL	Delinquent Indicator
avg_days_susp	0.3279	INTERVAL	Days Suspended Last 6M
times_susp	0.2773	NOMINAL	Number of Times Suspended
wrk_orders	0.2289	NOMINAL	Open Work Orders
pymts_late_ltd	0.1877	NOMINAL	Total Late Payments Lifetime
days_openwrkorders	0.0562	INTERVAL	Days of Open Work Orders
IMP_REP_MOU_ONNET_PCT_MOM	0.0488	INTERVAL	Imputed Replacement: Minutes On Network Pct Change Month over Month
IMP_REP_MB_DATA_NDIST_MO6M	0.0236	INTERVAL	Imputed Replacement: 6M Avg Billed Data Usage Normally Distributed
IMP_REP_MOU_TOTAL_PCT_MOM	0.0099	INTERVAL	Imputed Replacement: Minutes Total Pct Change Month over Month
times_delinq	0.0091	NOMINAL	Consecutive Mths Delinquent
REP_CALLS_TOTAL	0.0059	INTERVAL	Replacement: Total Calls Curr
LOG_MB_Data_Usg_M06	0	INTERVAL	Transformed MB of Data Usage Month 6

The most important input variables are listed in descending order of their importance. **Total Days Over Plan** appears to be the most important predictor, followed by **Handset Age Group**, and so on. Input relative importance is calculated by depth-one decision trees using each input to estimate the predicted values of the support vector machine model.

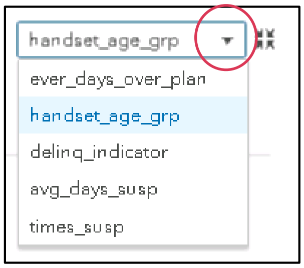
16. Click ⛶ to exit the maximized view of this window.

17. Click ⛶ to expand the Partial Dependence plot.



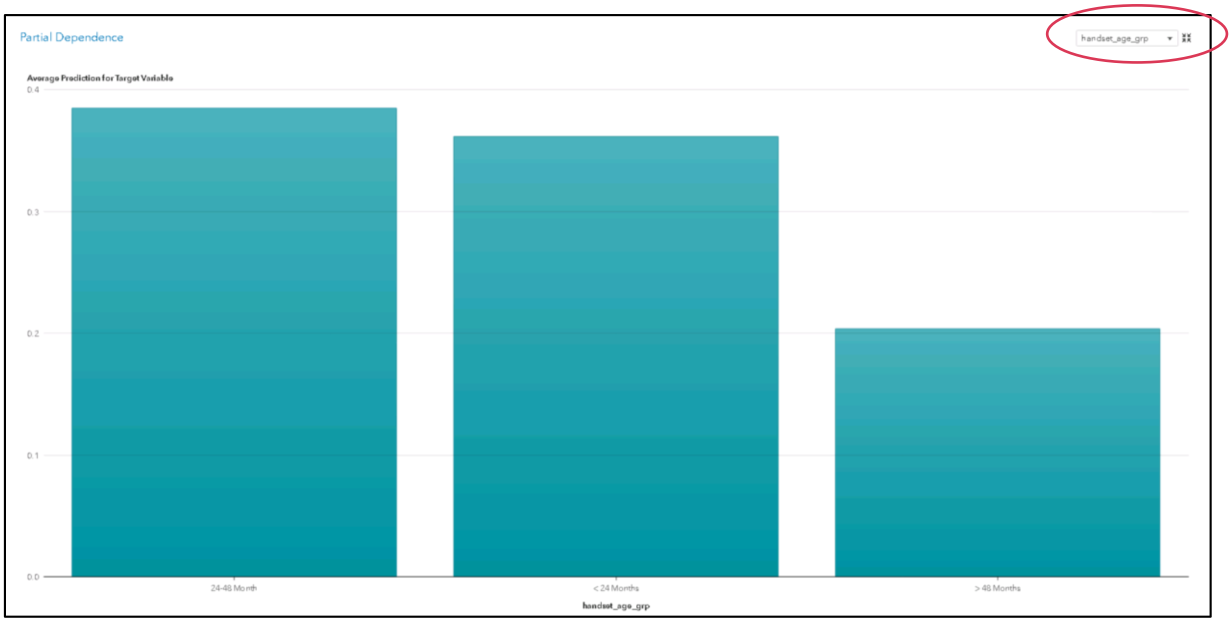
This plot shows the relationship between **Total Days Over Plan** and the model's prediction. There is a positive linear relationship. This is an important insight for the business.

18. To see the relationship between model's prediction and other variables, click the drop-down arrow in the upper right corner of the window.



These are the five most important inputs in the Input Relative Importance table.

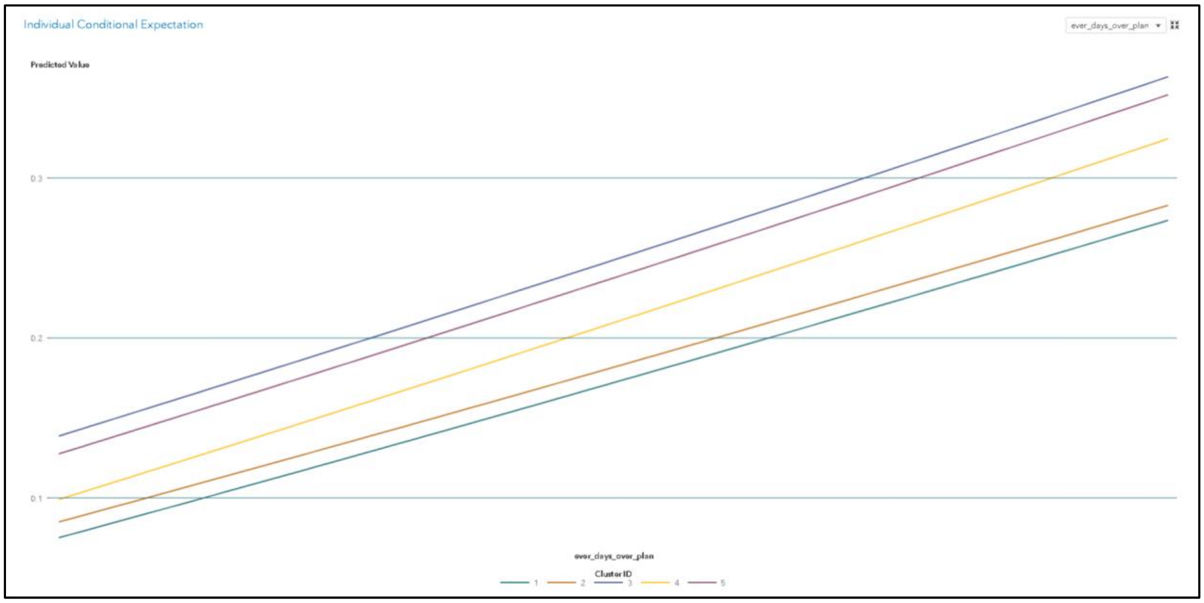
19. Select the **Handset Age Group** variable.



There is a decrease in the probability of churn the older the handset is. Does this make business sense?

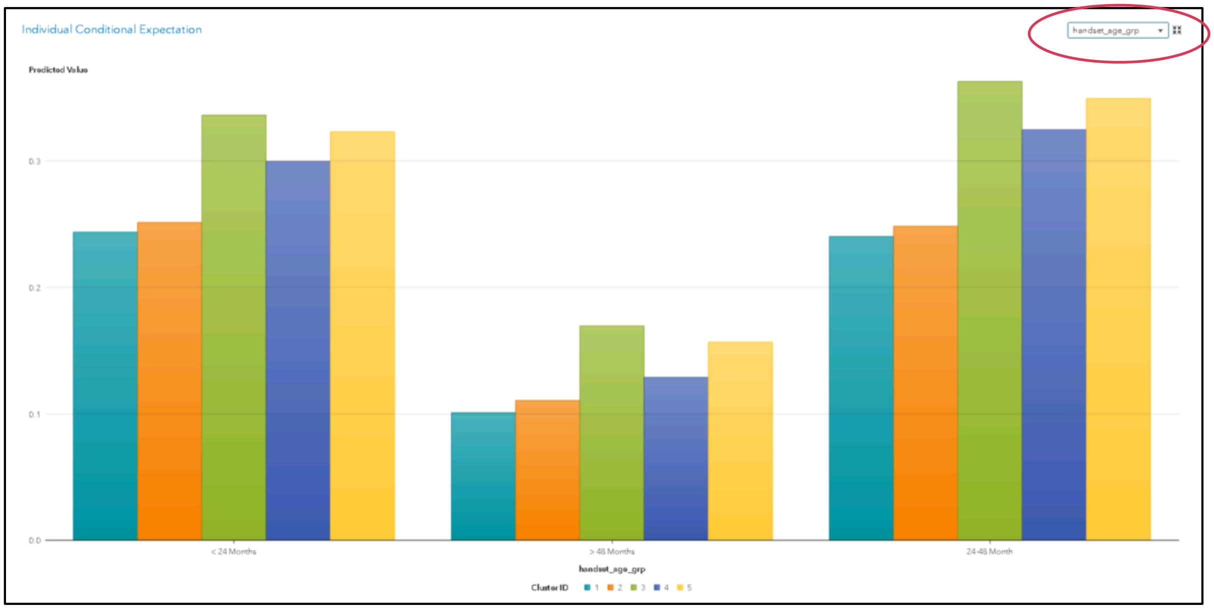
20. Click to exit the maximized view of this window.

21. Click ⛶ to expand the Individual Conditional Expectation plot.



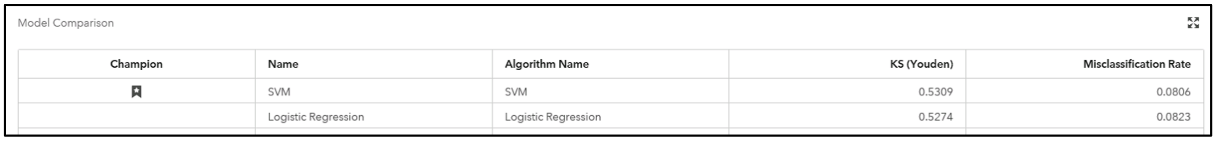
ICE plots help resolve interesting subgroups and interactions between model variables. The most useful feature to observe when evaluating an ICE plot of an interval input is intersecting slopes. Intersecting slopes indicate that there is an interaction between the plot variable and one or more complementary variables. **Total Days Over Plan** does not show any interactions**.**

22. Click the drop-down arrow in the upper right corner of the window to see ICE plots of other variables. Select the **Handset Age Group** variable.



A segmented ICE plot is created from a cluster of observations instead of on individual observations. The most useful feature to observe when evaluating an ICE plot of a categorical input is significant differences between each cluster's plot. Significant differences between each cluster's plot indicate group effects. For **Handset Age Group**, the differences in the plot indicate that there are significant differences between the clusters.

23. Click to exit the maximized view of this window.
24. Close the Results window.
25. Run the entire pipeline and view the results of model comparison. The Support Vector Machine model is the champion of this pipeline based on default KS.

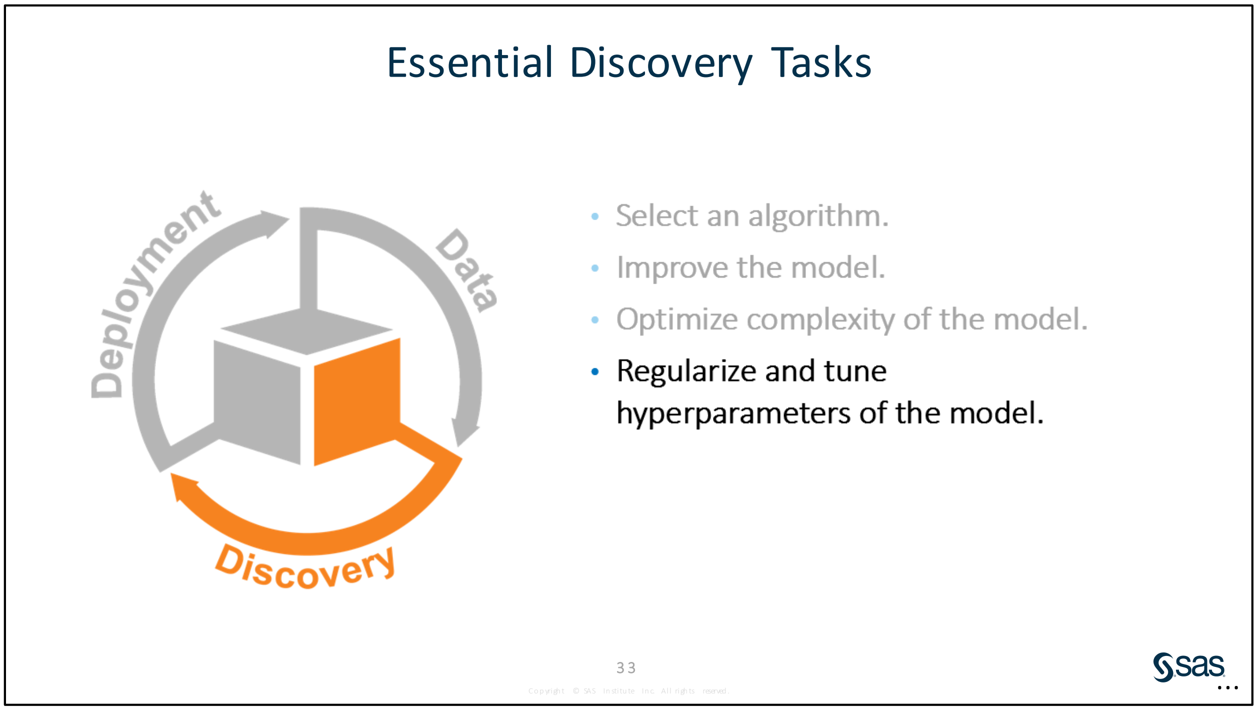


Model Comparison

Champion	Name	Algorithm Name	KS (Youden)	Misclassification Rate
🔖	SVM	SVM	0.5309	0.0806
	Logistic Regression	Logistic Regression	0.5274	0.0823

26. Close the Results window.

End of Demonstration



Essential Discovery Tasks

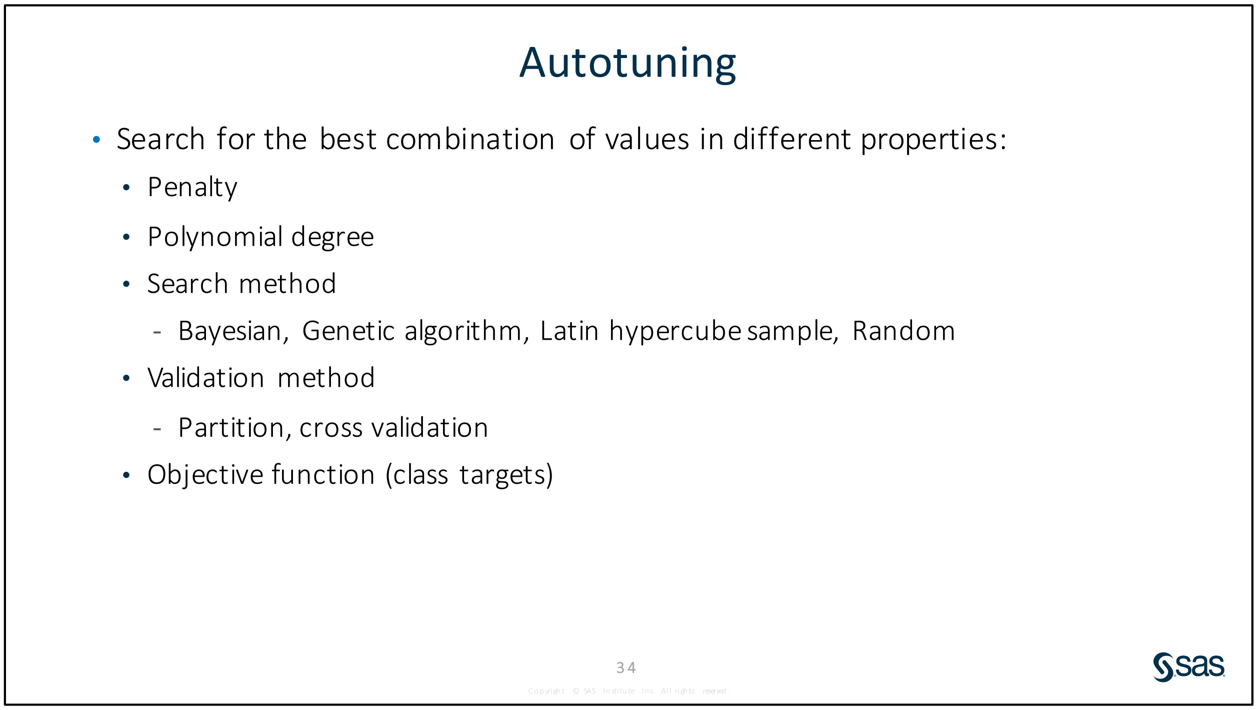
Deployment

Data

Discovery

- Select an algorithm.
- Improve the model.
- Optimize complexity of the model.
- Regularize and tune hyperparameters of the model.

One of the hardest processes in support vector machine models is to find the model parameters that minimize the loss function.



Autotuning

- Search for the best combination of values in different properties:
  - Penalty
  - Polynomial degree
  - Search method
    - Bayesian, Genetic algorithm, Latin hypercube sample, Random
  - Validation method
    - Partition, cross validation
  - Objective function (class targets)

When the autotuning feature is used, SAS Visual Data Mining and Machine Learning returns the penalty, the kernel function, and the degree of the kernel function if it is a polynomial function. Autotuning is invoked by selecting the **Performing Autotuning** option property on the Options tab in the Support Vector Machine node.

You can define the search method for the hyperparameters, as Bayesian, Genetic Algorithm, Latin hypercube sample, or Random sample. The genetic algorithm method uses an initial Latin hypercube sample that seeds a genetic algorithm to generate a new population of alternative configurations at each iteration. The Latin hypercube method performs an optimized grid search that is uniform in each tuning parameter, but random in combinations. The Random method generates a single sample of purely random configurations. The Bayesian method uses priors to seed the iterative optimization.

Finally, you can specify the number of tuning evaluations in one iteration. This option is available only if the Search method is Genetic algorithm or Bayesian. Similarly, you can specify the maximum number of tuning evaluations and the maximum number of tuning iterations.

For the search method Random or Latin hypercube is also possible to specify a sample size.

Finally, you can specify the validation method for finding the objective value, including partition and cross validation – including the proportion of the validation data set and the number of folds for cross validation – and the objective function depending on the level of the target variable.

Note: Performing autotuning can substantially increase run time.

Autotuning searches for the best combination of the following support vector machine parameters:

Number of Hidden Layers specifies whether to autotune the number of hidden layers. It ranges from 1 to 5. The default initial value is 1. The default range is from 0 to 2.

Number of Neurons specifies whether to autotune the number of neurons. It ranges from 1 to 1000. The default initial value is 1. The default range is from 1 to 100.

Penalty specifies whether to autotune the penalty value. The **initial value** is **1**. The search process can be ranged from **0.000001** to **100**, defined by **From** and **To** options.

Polynomial degree specifies whether to autotune the polynomial degree for the SVM model. The initial value is 1. The search process can be ranged from **1** to **3**, defined by From and To options.

Search Options specifies the options for autotuning searching. The following options are available:

- **Genetic algorithm** uses an initial Latin Hypercube sample that seeds a genetic algorithm. The Genetic algorithm generates a new population of alternative configurations at each iteration.
- **Latin hypercube sample** performs an optimized grid search that is uniform in each tuning parameter, but random in combinations.
- **Random** generates a single sample of purely random configurations.
- **Bayesian** uses priors to seed the iterative optimization.

Number of evaluations per iteration specifies the number of tuning evaluations in one iteration. This option is available only if the Search method is Genetic algorithm or Bayesian. The default value is 10. It ranges from 2 to 2,147,483,647.

Maximum number of evaluations specifies the maximum number of tuning evaluations. This option is available only if the Search method is Genetic algorithm or Bayesian. The default value is 50. It ranges from 3 to 2,147,483,647.

Maximum number of iterations specifies the maximum number of tuning iterations. This option is available only if the Search method is Genetic algorithm or Bayesian. The default value is 5. It ranges from 1 to 2,147,483,647.

Sample size specifies the sample size. This option is available only if the Search method is Random or Latin hypercube sample. The default value is 50. It ranges from 2 to 2,147,483,647.

There are some general options associated with the autotuning search.

Validation method specifies the validation method for finding the objective value. If your data is partitioned, then that partition is used. Validation method, Validation data proportion, and Cross validation number of folds are all ignored.

- **Partition** specifies using the partition validation method. With partition, you specify proportions to use for randomly assigning observations to each role.
  - **Validation data proportion** specifies the proportion of data to be used for the Partition validation method. The default value is 0.3.
- **K-fold cross validation** specifies using the cross validation method. In cross validation, each model evaluation requires k training executions (on k-1 data folds) and k scoring executions (on one holdout fold). This increases the evaluation time by approximately a factor of k.
  - **Cross validation number of folds** specifies the number of partition folds in the cross validation process (the k defined above). Possible values range from 2 to 20. The default value is 5.

Nominal target objective function specifies the objective function to optimize for tuning parameters for a nominal target. Possible values are average square error, area under the curve, F1 score, F0.5 score, gamma, Gini coefficient, Kolmogorov-Smirnov statistic, multi-class log loss, misclassification rate, root average squared error, and Tau. The default value is misclassification rate.

Interval target objective function specifies the objective function to optimize for tuning parameters for an interval target. Possible values are average squared error, mean absolute error, mean square logarithmic error, root average squared error, root mean absolute error, and root mean squared logarithmic error. The default value is average square error.

Exercises

1. Building a Support Vector Machine Model

a. Build a Support Vector Machine model using the Autotune feature. Add a Support Vector Machine node to the Chapter 5 pipeline, connected to the Variable Selection node. Use the Autotune feature. Explore the settings that are made available on when **Autotune** is selected. Run a few options by changing the range of the parameters search.

Note: This exercise might take several minutes to run.

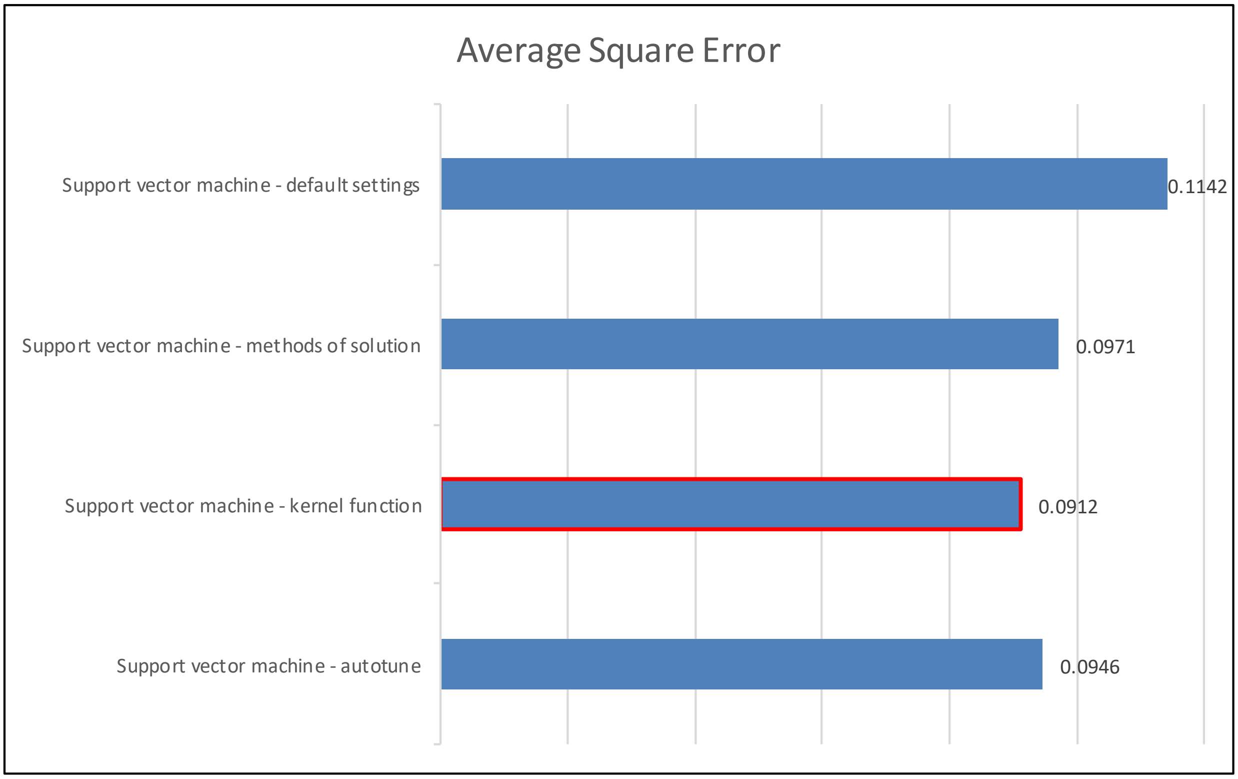
b. What kernel was selected during the autotune process? What is the value of the penalty parameter, and is it much different from the default value (1) used for the other SVMs?

c. How does the autotuned SVM compare to the other models in the pipeline? Consider statistics such as KS and misclassification rate.

End of Exercises

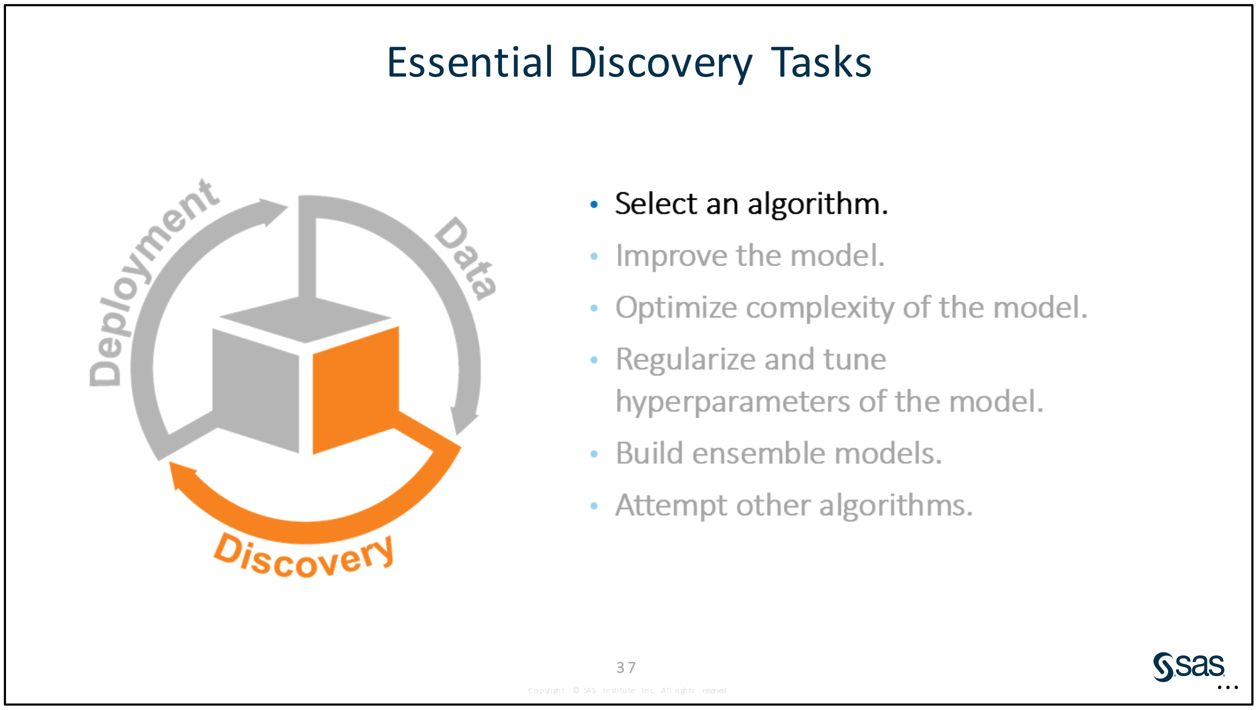
Summary of SVM Models

The following chart shows the different performances for the Support Vector Machine models. The Support Vector Machine model was improved based on the topics covered in this chapter, such as the introduction, the methods of solution, and the kernel functions. Starting from the default settings, each one of those topics were applied to improve the model's performance. Finally, the support vector machine based on the Autotune feature was created.



The tuned support vector machine based on the kernel functions achieved the best model, beating all other configurations, including the one based on the Autotune feature.

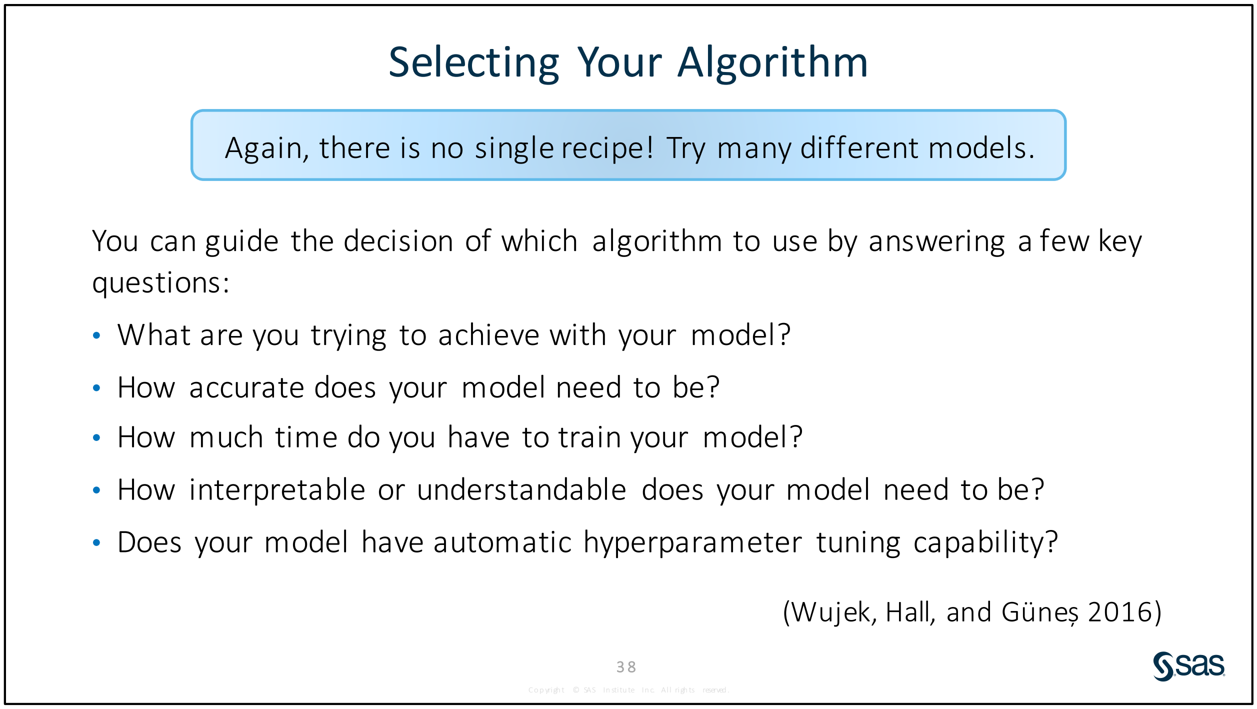
5.4 Selecting Your Algorithm



Essential Discovery Tasks

- Select an algorithm.
- Improve the model.
- Optimize complexity of the model.
- Regularize and tune hyperparameters of the model.
- Build ensemble models.
- Attempt other algorithms.

When you are presented with a data set, the first thing to consider is how to obtain results, no matter what those results might look like. So, let’s come back to selecting an algorithm.



Selecting Your Algorithm

Again, there is no single recipe! Try many different models.

You can guide the decision of which algorithm to use by answering a few key questions:

- What are you trying to achieve with your model?
- How accurate does your model need to be?
- How much time do you have to train your model?
- How interpretable or understandable does your model need to be?
- Does your model have automatic hyperparameter tuning capability?

(Wujek, Hall, and Güneș 2016)

Users with less experience tend to choose algorithms that are easy to implement and can obtain results quickly. This approach is acceptable if it is the first step of the process. After you obtain some results and become more familiar with the data, you might spend more time experimenting with more sophisticated algorithms. This might strengthen your understanding of the data, and potentially further improve the results.

Even in this stage, the best algorithms might not be the methods that have achieved the highest reported accuracy. Most algorithms usually require careful tuning and extensive training to obtain the best achievable performance. Selecting the modeling algorithm for your machine learning application can sometimes be the most difficult part. The decision of which algorithm to use can be guided by answering a few key questions (Wujek, Hall, and Güneș 2016):

What is the size and nature of your data? If you expect a linear relationship between your features and your target, linear or logistic regression or a linear kernel support vector machine might be sufficient. Linear models are also a good choice for large data sets due to their training efficiency and due to the curse of dimensionality. As the number of features increase, the distance between points grows and observations are more likely to be linearly separable. To an extent, nonlinearity and interaction effects can be captured by adding higher-order polynomial and interaction terms in a regression model. More complex relationships can be modeled through the power of the more sophisticated machine learning algorithms such as decision trees, random forests, neural networks, and nonlinear kernel support vector machines. Of course, these more sophisticated algorithms can require more training time and might be unsuitable for very large data sets.

What are you trying to achieve with your model? Are you creating a model to classify observations, predict a value for an interval target, detect patterns or anomalies, or provide recommendations? Answering this question will direct you to a subset of machine learning algorithms that specialize in the problem.

How accurate does your model need to be? Although you always want your model to be as accurate as possible when applied to new data, it is still always good to strive for simplicity. Simpler models train faster and are easier to understand, making it easier to explain how and why the results were achieved. Simpler models are also easier to deploy. Start with a regression model as a benchmark, and then train a more complex model such as a neural net, random forest, or gradient boosted model. If your regression model is much less accurate than the more complex model, you have probably missed some important predictor or interaction of predictors. An additional benefit of a simpler model is that it will be less prone to overfitting the training data.

How much time do you have to train your model? This question goes together with the question of how accurate your model needs to be. If you need to train a model in a short amount of time, linear or logistic regression and decision trees are probably your best options. If training time is not an issue, take advantage of the powerful algorithms (neural networks, support vector machines, gradient boosting, and so on) that iteratively refine the model to better represent complex relationships between features and the target of interest.

How interpretable or understandable does your model need to be? It is very important to establish the expectations of your model consumer about how explainable your model must be. If an uninterpretable prediction is acceptable, you should use as sophisticated an algorithm as you can afford in terms of time and computational resources. Train a neural network, a support vector machine, or any flavor of ensemble model to achieve a highly accurate and generalizable model. If interpretability or explainable documentation is important, use decision trees or a regression technique, and consider using penalized regression techniques, generalized additive models, quantile regression, or model averaging to refine your model.

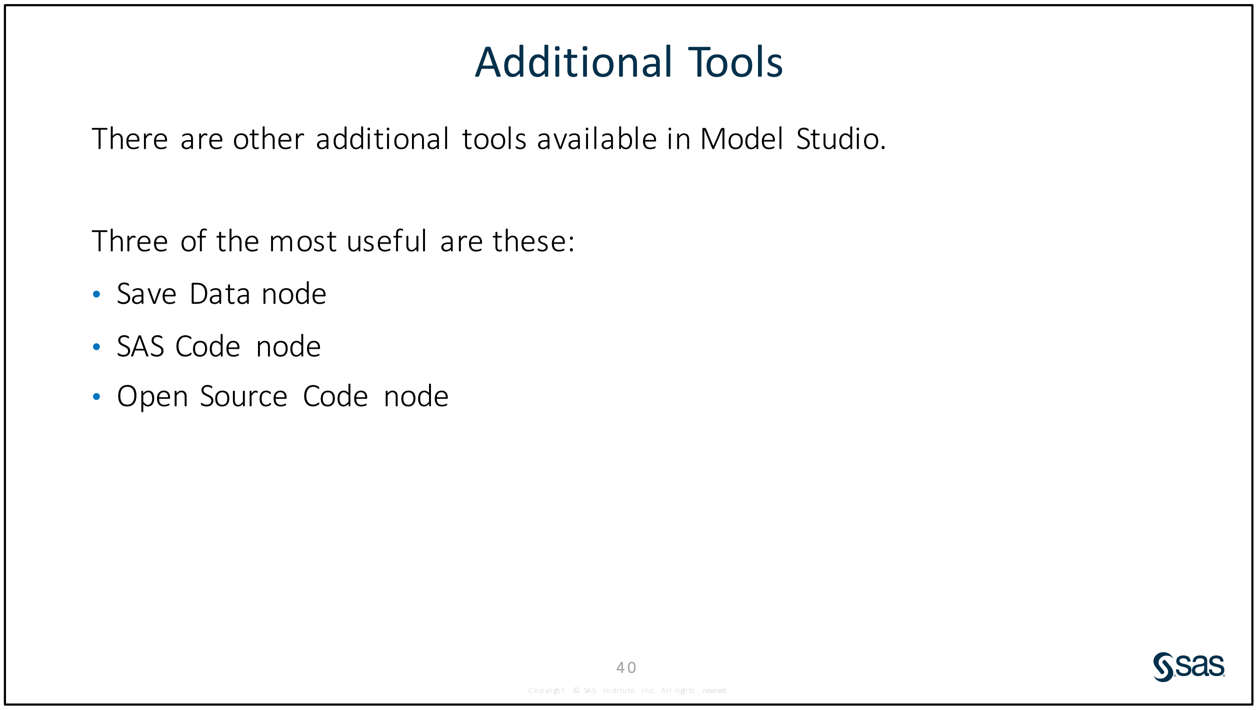
Does you model have automatic hyperparameter tuning capability? Optimal hyperparameter settings are extremely data-dependent. Therefore, it is difficult to offer a general rule about how to identify a subset of important hyperparameters for a learning algorithm or how to find optimal values of each hyperparameter that would work for all data sets. Controlling hyperparameters of a learning algorithm is very important because proper control can increase accuracy and prevent overfitting.

The following table presents some best practices for selecting SAS Visual Data Mining and Machine Learning supervised learning algorithms:

Algorithm Type	Target Type	Usage	Scale	Interpre-tability	Auto-tuning	Common Concerns
Regression (Linear, Logistic, GLM)	• Linear regression and GLM for interval target. • Logistic regression for nominal and binary target.	• Modeling linear or linearly separable phenomena. • Manually specifying nonlinear and explicit interaction. • LASSO regression includes a regularization term for linear and logistic regression to deal w ith multicollinearity and overfitting issues.	Small to large data sets	High	No	• Missing values • Outliers • Standardization • Parameter tuning
SVM	Binary	Modeling linear or linearly separable phenomena by using linear kernels or polynomial kernels up to degree three	Small to large data sets	Low	Yes	• Missing values • Overfitting • Outliers • Standardization • Parameter tuning
Tree-based Modeling (Decision Tree, Forest, Gradient Boosting)	• Interval • Binary • Nominal	• Modeling nonlinear and nonlinear separable phenomena in large data sets • Interactions considered automatically, but implicitly • Missing values and outliers in input variables handled automatically in many implementations • Tree ensembles (forests, gradient boosting) can increase prediction accuracy and decrease overfitting, but also decrease scalability and interpretability	Mid-size to large data sets	Moderate	Yes	• Instability w ith small training sets • Gradient boosting can be unstable w ith noise or outliers • Overfitting • Parameter tuning

Algorithm Type	Target Type	Usage	Scale	Interpre-tability	Auto-tuning	Common Concerns
Neural Network	• Interval • Binary • Nominal	• Modeling nonlinear and nonlinearly separable phenomena. • All interactions considered in fully connected, multilayer topologies.	Mid-size to large data sets	Low	Yes	• Missing values • Overfitting • Outliers • Standardization • Parameter tuning
Bayesian Network	• Binary • Nominal	• Modeling linearly separable phenomena in large data sets. • Well suited for extremely large data sets where complex methods are intractable.	Small to extre-mely large data sets	Moderate	No	• Linear independence assumption • Infrequent categorical levels

5.5 Additional Tools



Additional Tools

There are other additional tools available in Model Studio.

Three of the most useful are these:

- Save Data node
- SAS Code node
- Open Source Code node

40

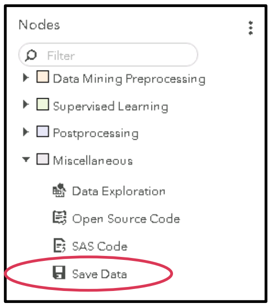
There are several additional tools available in Model Studio that have not been covered in previous chapters of this course. This section discusses three of these tools: the Save Data node, the SAS Code node, and the Open Source Code node.

Here are some other useful tools:

- **Batch Code**: The Batch Code node is a Supervised Learning node. It enables you to import external SAS models that are saved in batch code format.
- **Score Code import**: The Score Code Import node is a Supervised Learning node that enables you to import external models that are saved as SAS score code.
- **Ensemble**: The Ensemble node is a Postprocessing node. It creates new models by combining the posterior probabilities (for class targets) or the predicted values (for interval targets) from multiple predecessor models.

Save Data Node

The Save Data node is used to save data exported by a node in a pipeline to a CAS library.



41

The Save Data node is a Miscellaneous node that enables you to save the training table that is produced by a predecessor node to a caslib. This table could be partitioned into training, validation, or test sets based on the project settings. In that case, the table contains the **_partind_** variable that identifies the partitions.

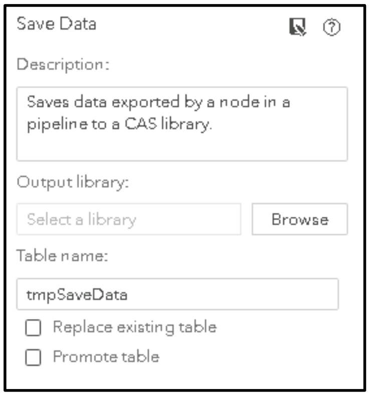
By default, the training table produced by a pipeline is temporary and exists only for the duration of the run of a node and has local session scope. The Save Data node enables you to save that table to disk in the location associated with the specified output library. This table can then be used later by other applications for further analysis or reporting.

The default output caslib where tables are to be saved can be specified in the Output Library Project settings. You can overwrite this location using the Output library property.

In addition, you can load the table in memory and promote this table to have global scope in the specified caslib. This enables multiple CAS sessions to access this table.

If you run the node, the results consist of an output table containing information about the saved table, including a list of variables and their basic attributes.

Properties Panel



Save Data

Description:

Saves data exported by a node in a pipeline to a CAS library.

Output library:

Select a library

Table name:

tmpSaveData

□ Replace existing table

□ Promote table

The properties of the Save Data node are as follows:

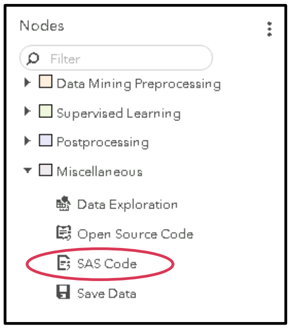
- **Output library** – specifies the output caslib where the table will be saved on disk. Use **Browse** to navigate to the proper library. If the user has specified an output library under **Project Settings**, then this library will be used by default.
- **Table name** – specifies the name for the CAS table being saved. The default value is **tmpSaveData**.
- **Replace existing table** – specifies whether to override an existing CAS table with the same name when saving. By default, this option is deselected.
- **Promote table** – specifies whether to load the table in memory and promote the table to global space. By default, this option is deselected.

After running the node, you can open the Results window. Two tabs are in the Results window: Properties and Output.

- **Properties** – specifies the various properties selected before running the node. These include the output library, the table name, whether to replace or promote the table, and the CAS session ID.
- **Output** – displays the SAS output of the saved data run.

SAS Code Node

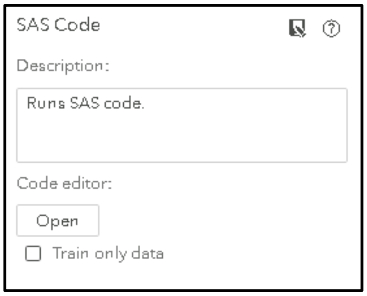
The SAS Code node is used to run SAS code.



The SAS Code node is a Miscellaneous node that enables you to incorporate new or existing SAS code into Model Studio pipelines. The node extends the functionality of Model Studio by making other SAS procedures available for use in your data mining analysis. You can also write SAS DATA steps to create customized scoring code, conditionally process data, or manipulate existing data sets. The SAS Code node is also useful for building predictive models, formatting SAS output, defining table and plot views in the user interface, and for modifying variables' metadata. The node can be placed at any location within a pipeline (except after the Ensemble or Model Comparison nodes). By default, the SAS Code node does not require data. The exported data that is produced by a successful SAS Code node run can be used by subsequent nodes in a pipeline.

To indicate that the SAS Code node produces a model that should be assessed, right-click the **SAS Code** node and select **Move ⇨ Supervised Learning**. If a SAS Code node that is marked as a Supervised Learning node does not generate any score code, either as DS1 or as an analytical store (astore), then no assessment reports nor model interpretability reports are generated. If the node produces score code that does not create the expected predicted or posterior probability variables, then the node will fail.

Properties Panel


SAS Code

Description:

Runs SAS code.

Code editor:

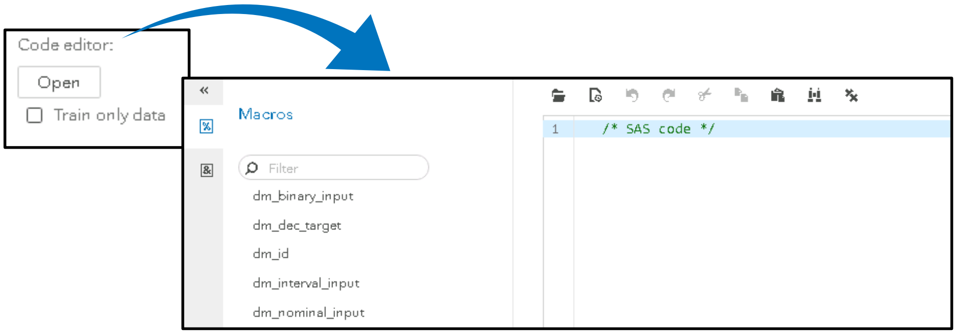
□ Train only data

The SAS Code node properties are as follows:

- **Code editor** – invokes the SAS Code Editor.
- **Train only data** – specifies whether the node should receive the training observations only if the data is partitioned. By default, this option is deselected. Currently, this property is unavailable for this node. To specify that the node receive only training data, add the following WHERE clause to your code:

```
where &dm_partitionvar.=1;
```

Code Editor



```
/* SAS code */
```

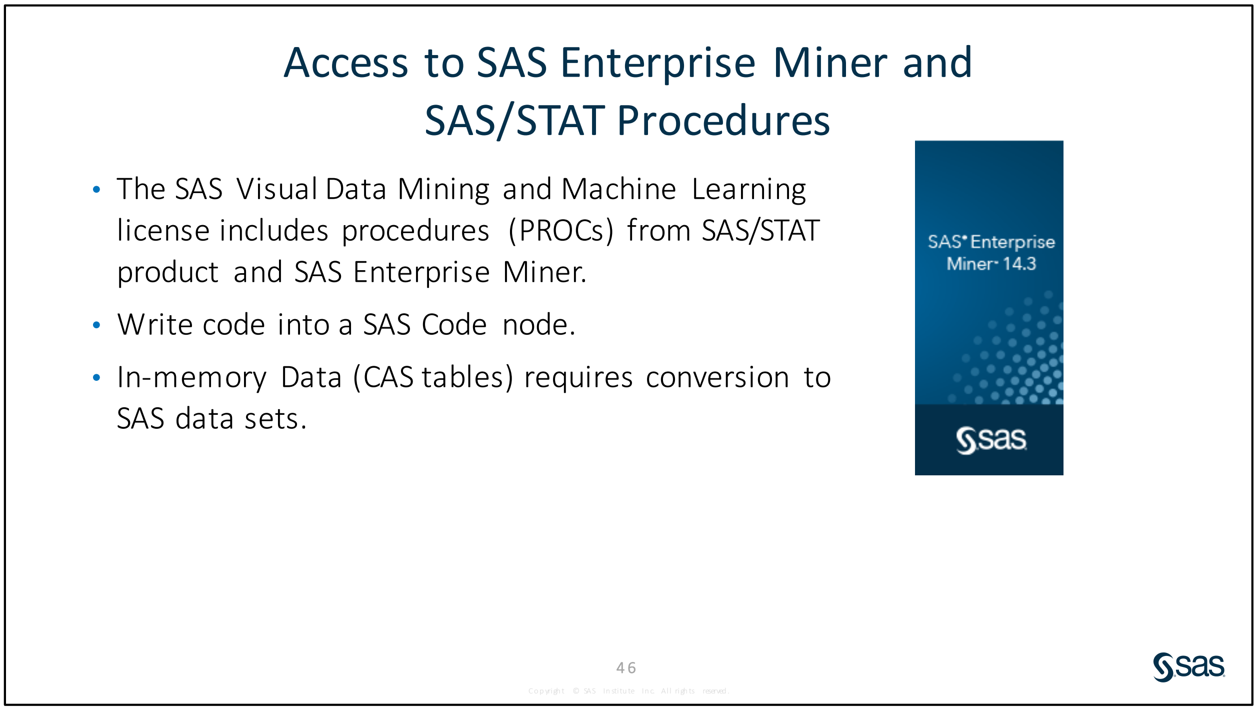
45

The code editor window is opened from a property in the properties panel. The code editor window enables the user to view a Macros table and a Macro Variable table from the left column, which contain a list of macros and macro variables, respectively, that are available to the SAS session.

Additional options are available as shortcut buttons on the top of the editor window. These options enable you to do the following:

- browse
- control settings, which include general (such as showing line numbers and font size) and editing (such as enabling autocomplete and auto indention) code options
- undo and redo
- cut, copy, and paste
- find and replace
- clear all code

User-written code is saved through a shortcut button (the good old 3.5-inch floppy disk icon) in the upper right corner of the editor window.



Access to SAS Enterprise Miner and SAS/STAT Procedures

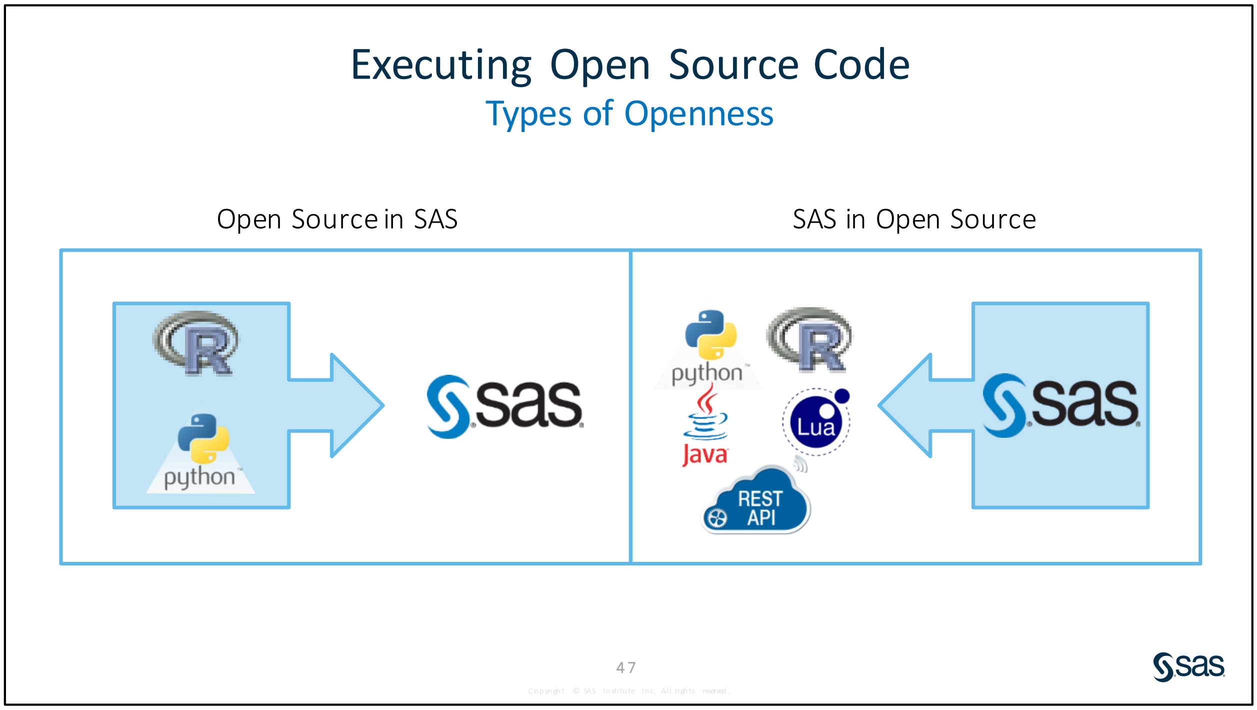
- The SAS Visual Data Mining and Machine Learning license includes procedures (PROCs) from SAS/STAT product and SAS Enterprise Miner.
- Write code into a SAS Code node.
- In-memory Data (CAS tables) requires conversion to SAS data sets.

SAS® Enterprise Miner® 14.3
sas

46

SAS Viya users have access to more power than they might realize. All SAS Enterprise Miner and SAS/STAT procedures are included with a Visual Data Mining and Machine Learning license on SAS Viya. This means that by using the SAS Code node in a pipeline, users have access to the Enterprise Miner procedures that are specific to that product and to the entire suite of tools available with SAS®9 SAS/STAT software.

The in-memory CAS table would require being copied to a location accessible by these procedures in the form of a SAS data set.



Executing Open Source Code

Types of Openness

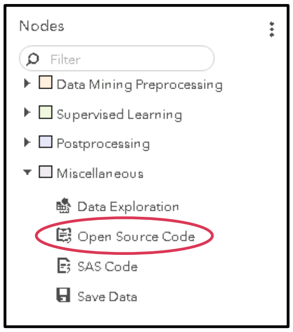
Open Source in SAS

SAS in Open Source

47

Open source in SAS Viya supports Python and R languages and requires Python or R and necessary packages to be installed on the same machine as the SAS Compute Server. It downloads data samples from SAS Cloud Analytic Services for use in Python or R code and transfers data by using a data frame or CSV file using the Base SAS Java Object.

Open Source Code Node



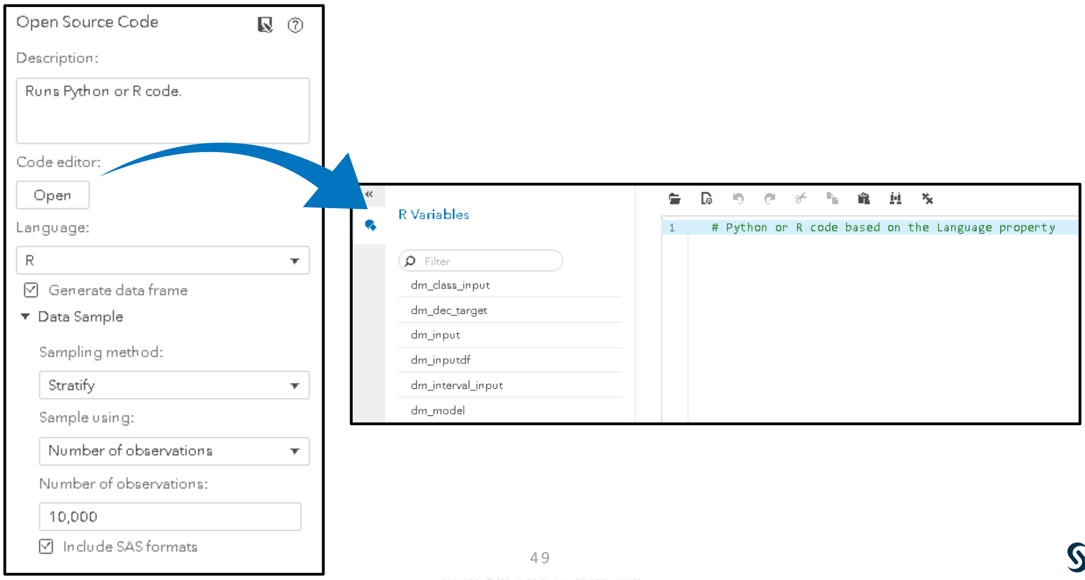
- The Open Source Code node is used to run Python or R code in a pipeline.
- Requires Python or R and necessary packages to be installed on the same machine as the SAS Compute Server.
- Cannot be part of an ensemble.
- Does not support registering, publishing, or downloading scoring code or scoring APIs.
- Enables the comparison of Python or R models within a Model Studio pipeline.

48

The Open Score Code node enables you to import external code that is written in Python or R. The version of Python or R software does not matter to the node, so any version can be used as the code is passed along. The Python or Rscript executable must be in system path on Linux, or the install directories can be specified with PYTHONHOME or RHOME on Windows.

The node enables the user to prototype machine learning algorithms that might exist in open source languages but have not yet been vetted to be included directly as a node in Model Studio. This node can subsequently be moved to a Supervised Learning group if a Python or R model needs to be assessed and included to be part of model comparison. The node can execute Python or R software regardless their versions.

Properties Panel



Open Source Code

Description:

Runs Python or R code.

Code editor:

Open

Language:

R

☑ Generate data frame

▾ Data Sample

Sampling method:

Stratify

Sample using:

Number of observations

Number of observations:

10,000

☑ Include SAS formats

R Variables

Filter

dm_class_input

dm_dec_target

dm_input

dm_inputdf

dm_interval_input

dm_model

```
# Python or R code based on the Language property
```

After selecting the language (Python or R) from properties, use the Open button to enter respective code in the editor. Because this code is not executed in CAS, a data sample (10,000 observations by default) is created and downloaded to avoid movement of large data. Use Data Sample properties to control the sample size and method. Apply caution and do not specify full data or a huge sample when the input data is large. When performing model comparison with other Supervised Learning nodes in the pipeline, note that this node might not be using full data.

Input data can be accessed by the Python or R code via a CSV (comma-separated-value file) or as a data frame. When **Generate data frame** is selected, a data frame is generated from the CSV, and input data is available in dm_inputdf, which is a pandas data frame in Python or an R data frame. When data are partitioned, an additional data frame, dm_traindf, is also available in the editor. That frame contains training data. If a Python or R model is built and needs to be assessed, corresponding predictions or posterior probabilities should be made available in **dm_scoreddf** data frame. To do so, right-click and select **Move** ⇨ **Supervised Learning** to indicate that model predictions should be merged with input data and model assessment should be performed. Note that the number of observations in dm_inputdf and dm_scoreddf should be equal for successful merge to occur.

Note that this node cannot support operations such as **Download score code**, **Register models**, **Publish models**, and **Score holdout data** from the Pipeline Comparison tab because it does not generate SAS score code.

Properties of the Open Source Code node are as follows:

- **Code editor** – invokes the SAS Code Editor.
- **Language** – specifies the open source language to be used. Available options for this property are R and Python. The default setting is R.
- **Generate data frame** – specifies whether to generate an R data frame or a pandas data frame in Python. In addition, categorical inputs are encoded as factors in R. If this option is disabled, the input data should be accessed as a CSV file. By default, this option is enabled.

- **Data Sample** – controls sampling of the data. By default, this property is collapsed. Thee Data Sample property has been expanded in the screen capture above. When expanded, the subcategories are shown. The subcategories are as follows:
  - **Sampling Method** – specifies the sampling method. When the input data has a partition variable or a class target (or both), the sample is stratified using them. Otherwise, a simple random sample is used. The available settings are None, Simple Random and Stratify. The default setting is Stratify.
  - **Sample using** – specifies whether to sample using the number of observations or the percent of observations from input data. The available settings are Number of observations and Percent of Observations. The default setting is Number of observations.
  - **Number of Observations** or **Percent of Observations** –depends on the setting for the **Sample using** property. When **Sample using** is set to **Number of Observations**, this property specifies the number of observations to sample from input data. The default in this case is 10,000, and the user can enter numeric values manually. When **Sample using** is set to **Percent of Observations**, this property specifies the percent of observations to sample from input data. In this case, a slider bar appears which ranges from 1 to 100 and the default setting is 10.
  - **Include SAS formats** – specifies whether to include SAS formats in input data to downloaded CSV files, when passing data to open source software. By default, this option is enabled.

Like the SAS Code node, for the Open Source Code node, the code editor window is opened from a property in the properties panel. The code editor window enables the user to view a list of R variables or Python variables, depending on what open source language is being used, that are available to the editor session.

Additional options are available as shortcut buttons that are identical to those described earlier in this section of the SAS Code node.

Further information on the Open Source Code node in Model Studio, including a short video illustrating use of the node, can be found here: https://communities.sas.com/t5/SAS-Communities-Library/How-to-execute-Python-or-R-models-using-the-Open-Source-Code/ta-p/499463

5.6 Solutions

Solutions to Exercises

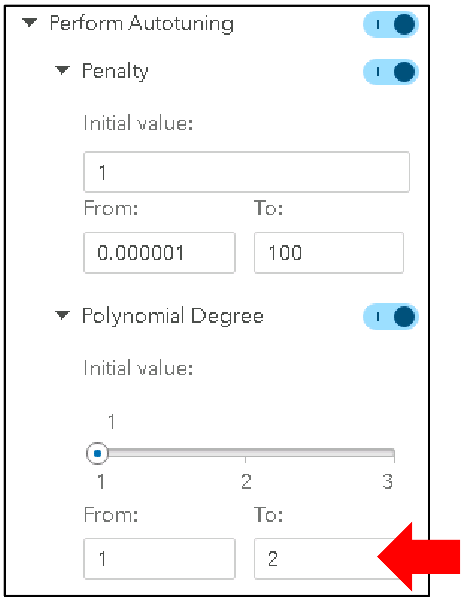
1. Building a Support Vector Machine Model

a. Build a support vector machine using the Autotune feature. Add an SVM node to the Chapter 5 pipeline, connected to the Variable Selection node. Use the Autotune feature. Explore the settings that are made available when **Autotune** is selected, but keep all properties at their defaults, ***except the polynomial degree***.

1) On the Starter Template pipeline, right-click the **Variable Selection** node and select **Add below ⇨ Supervised Learning ⇨ SVM**.

2) In the properties pane, turn on the **Perform Autotuning** option. The default properties show starting values and ranges that are tried for each property in the SVM model.

3) Under **Polynomial Degree**, change the maximum range by changing **To** from **3** to **2**.



Perform Autotuning

Penalty

Initial value: 1

From: 0.000001 To: 100

Polynomial Degree

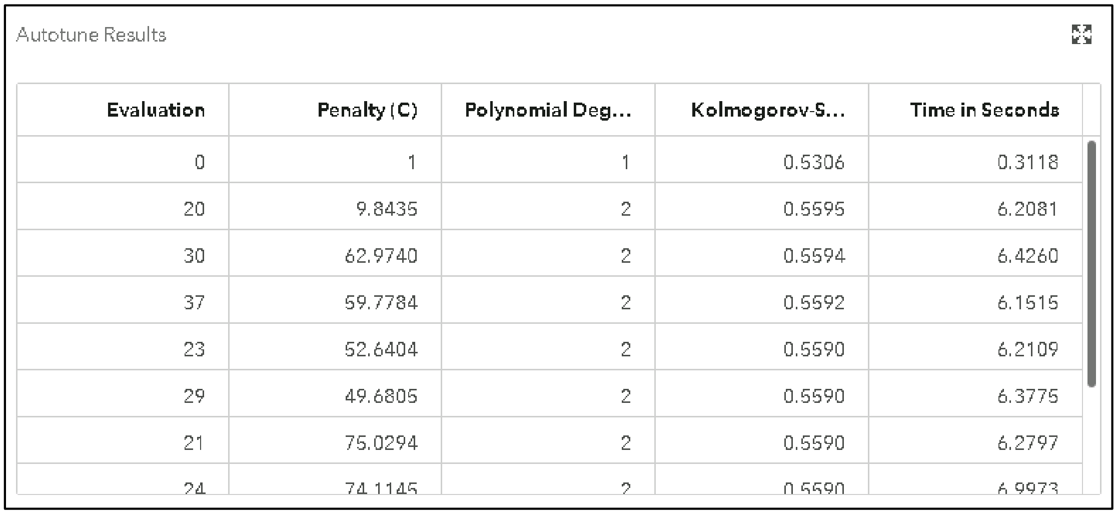
Initial value: 1

From: 1 To: 2

4) Right-click the **SVM** node and select **Run**. This process might take few minutes.

5) When the execution is over, right-click the **SVM** node and select **Results**.

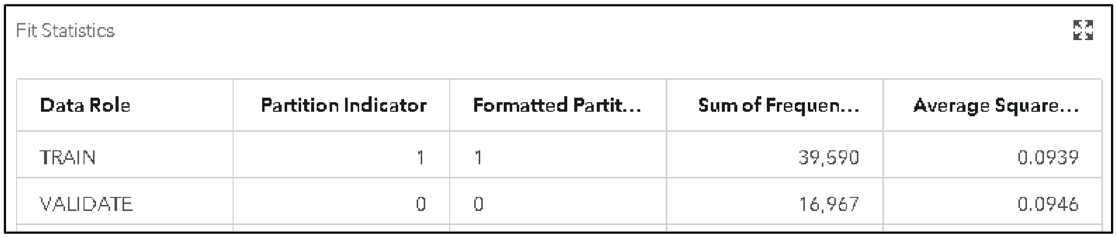
6) Examine the Results window. Maximize the Autotune Results window and notice the different evaluations performed. Restore the Autotune Results window.



Autotune Results

Evaluation	Penalty (C)	Polynomial Deg...	Kolmogorov-S...	Time in Seconds
0	1	1	0.5306	0.3118
20	9.8435	2	0.5595	6.2081
30	62.9740	2	0.5594	6.4260
37	59.7784	2	0.5592	6.1515
23	52.6404	2	0.5590	6.2109
29	49.6805	2	0.5590	6.3775
21	75.0294	2	0.5590	6.2797
24	74.1145	2	0.5590	6.9973

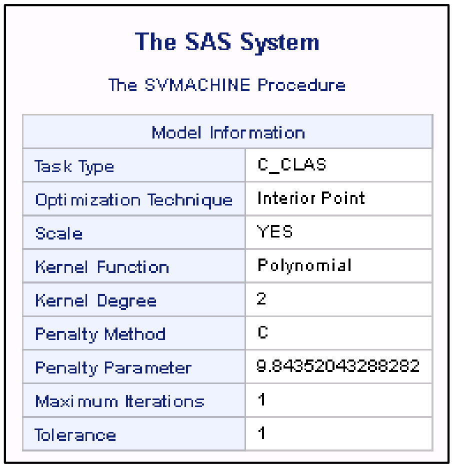
7) Scroll down and observe the Fit Statistics window. The average square error for the Autotune model is 0.0946 on the VALIDATE partition.



Fit Statistics

Data Role	Partition Indicator	Formatted Partit...	Sum of Frequen...	Average Square...
TRAIN	1	1	39,590	0.0939
VALIDATE	0	0	16,967	0.0946

8) Scroll down and maximize the Output window. This output shows the set of parameters selected for the final Support Vector Machine model.



The SAS System

The SVMACHINE Procedure

Model Information	
Task Type	C_CLAS
Optimization Technique	Interior Point
Scale	YES
Kernel Function	Polynomial
Kernel Degree	2
Penalty Method	C
Penalty Parameter	9.84352043288282
Maximum Iterations	1
Tolerance	1

b. What Kernel was selected during the autotune process? What is the value of the penalty parameter and is it much different from the default value (1) used for the other SVMs?

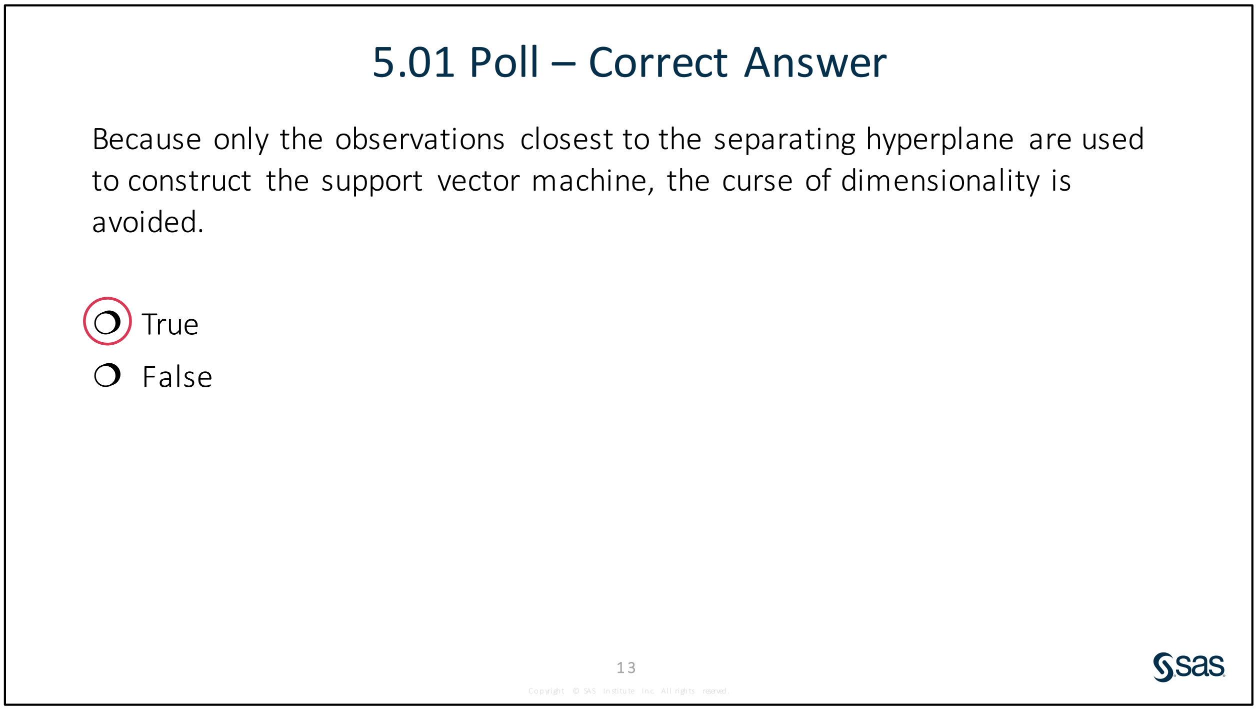
A polynomial kernel with degree of 2 was selected. The penalty term is 9.8435, which is very different from the default value of 1. One single iteration was selected.

c. How does the autotuned SVM compare to the other models in the pipeline? Consider statistics such as KS and misclassification rate.

The autotuned SVM was worse than the last model tuned during the demonstrations.

End of Solutions

Solutions to Student Activities (Polls/Quizzes)



5.01 Poll – Correct Answer

Because only the observations closest to the separating hyperplane are used to construct the support vector machine, the curse of dimensionality is avoided.

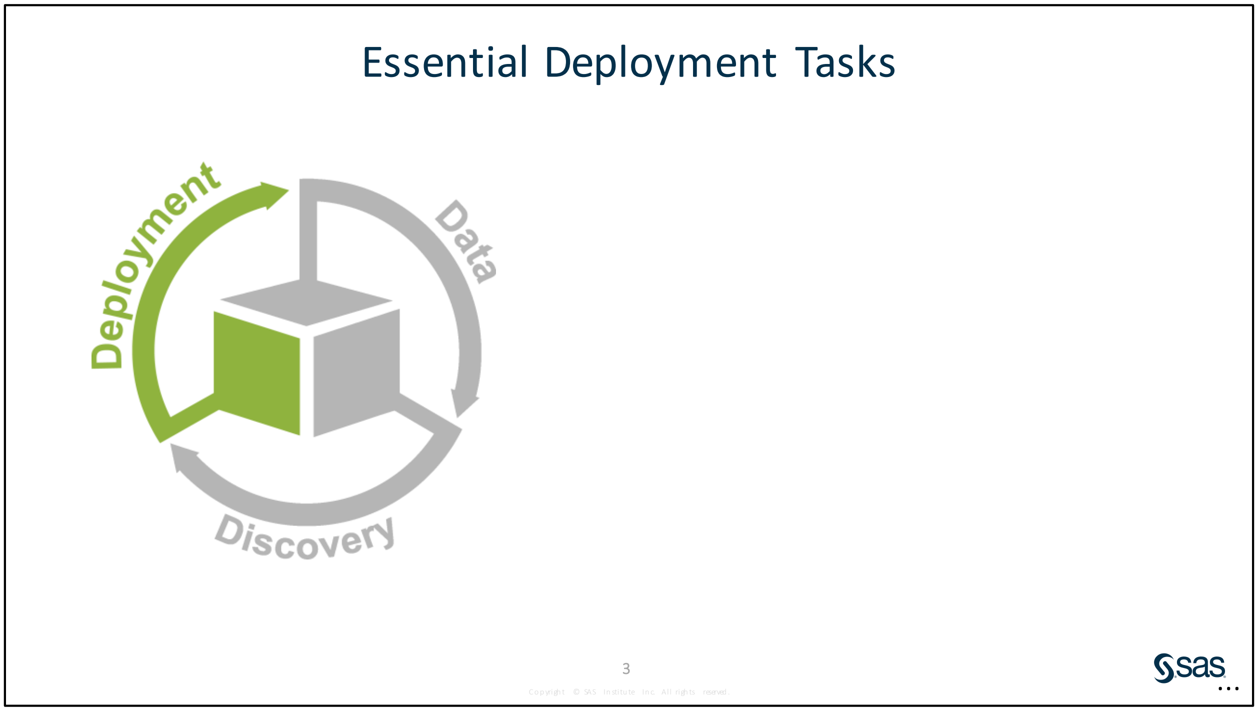
[✓ ❍ True]

❍ False

13

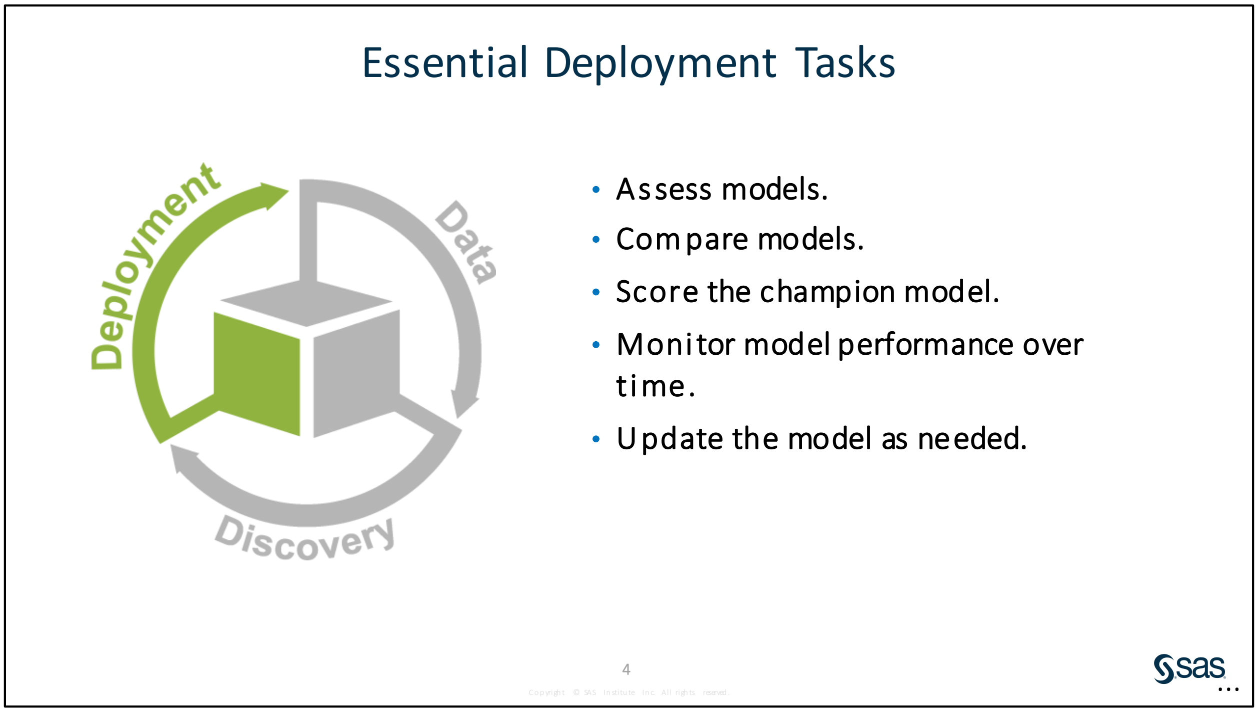
Chapter 6 Model Assessment and Deployment

6.1 Model Assessment and Comparison



Essential Deployment Tasks

Given that machine learning models tend to be difficult to interpret, their primary use is to create predictions that create value (monetary or otherwise) for an organization or other entity. The actual mechanism by which machine learning models create their predictions requires thought and attention. For example, making predictions on an individual's local machine is a good idea only for a limited time in most cases. If a model is useful, it needs to be used by an organization in an operational manner to make decisions quickly, if not automatically. Keep in mind that some level of data preparation has likely been applied to the data set in its original, raw form, and this must be accounted for when making predictions on new observations. Moving the logic that defines all the necessary data preparation and mathematical expressions of a sophisticated predictive model from a development environment such as a personal computer into an operational database is one of the most difficult and tedious aspects of machine learning. Mature, successful organizations are masters of this process—called *model deployment*, *deployment*, or *model production*.

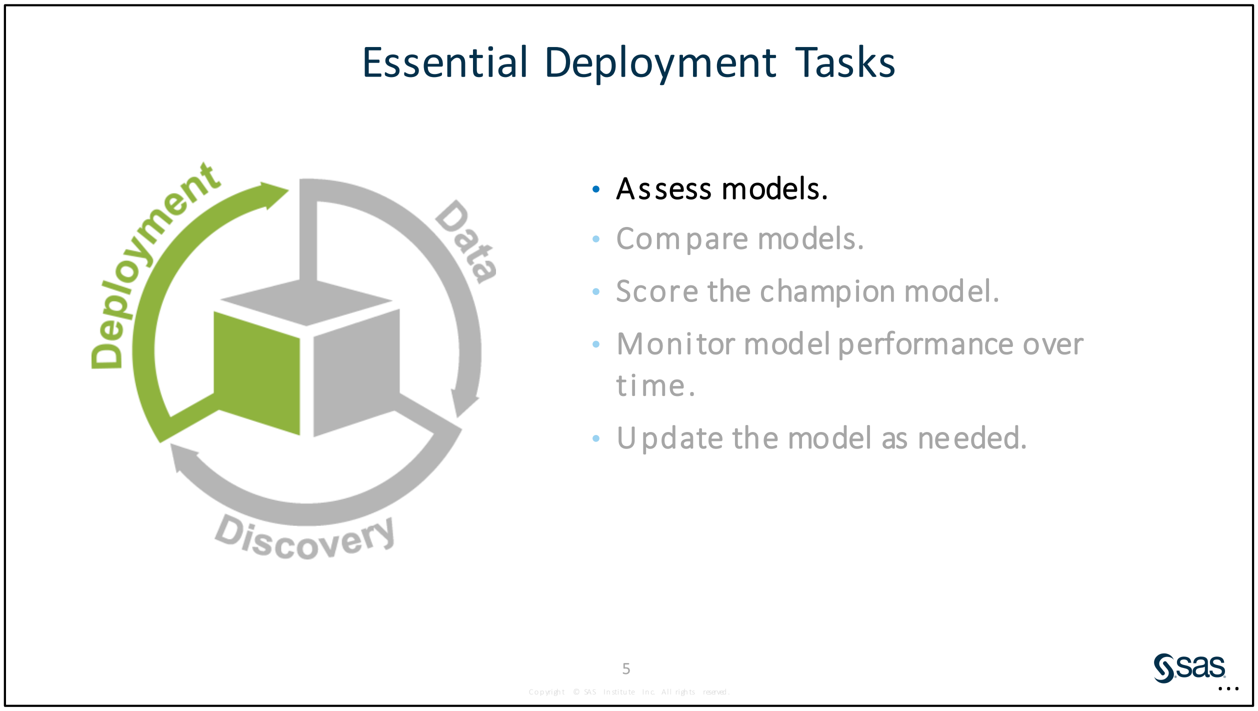


Essential Deployment Tasks

- Assess models.
- Compare models.
- Score the champion model.
- Monitor model performance over time.
- Update the model as needed.

4

We typically build several models, and it is therefore important first to assess individual models and then compare those several models and determine the best model typically called as a *champion* model. The champion model is then deployed into production, a process called *scoring*. Even after a model has been deployed, it must be monitored and then updated per requirements.

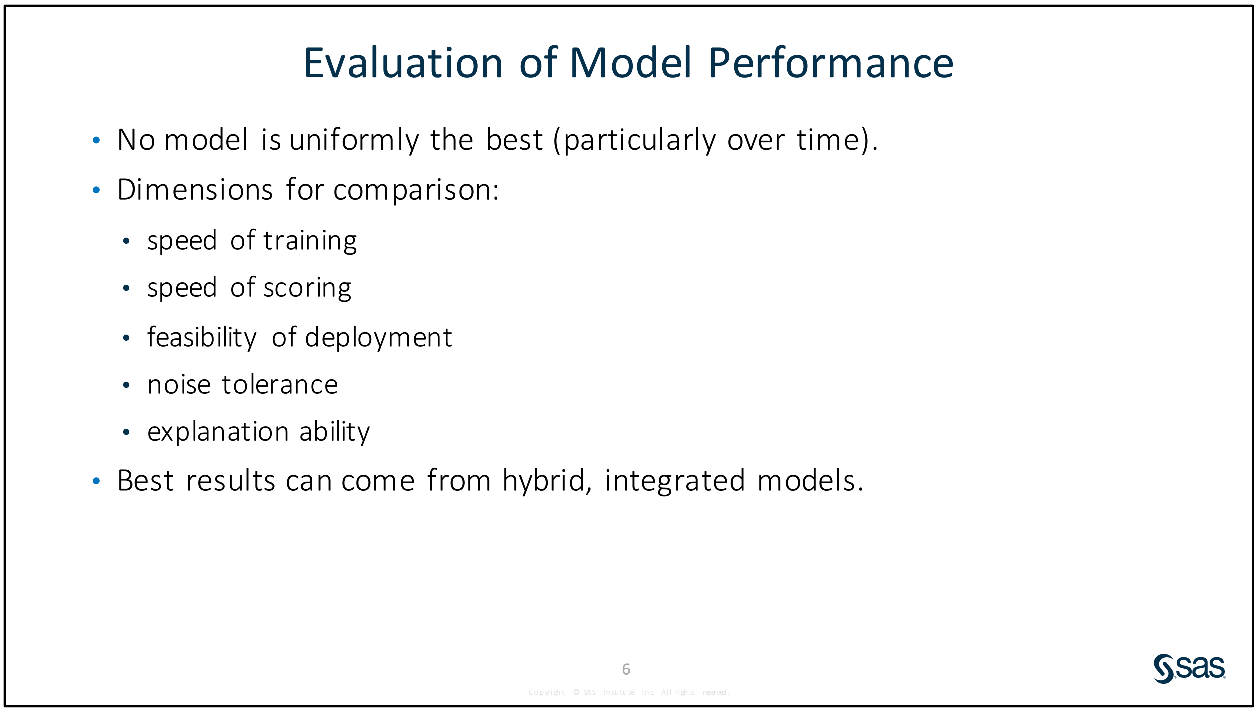


Essential Deployment Tasks

- Assess models.
- Compare models.
- Score the champion model.
- Monitor model performance over time.
- Update the model as needed.

5

Model assessment is evaluating the efficacy of your models built.



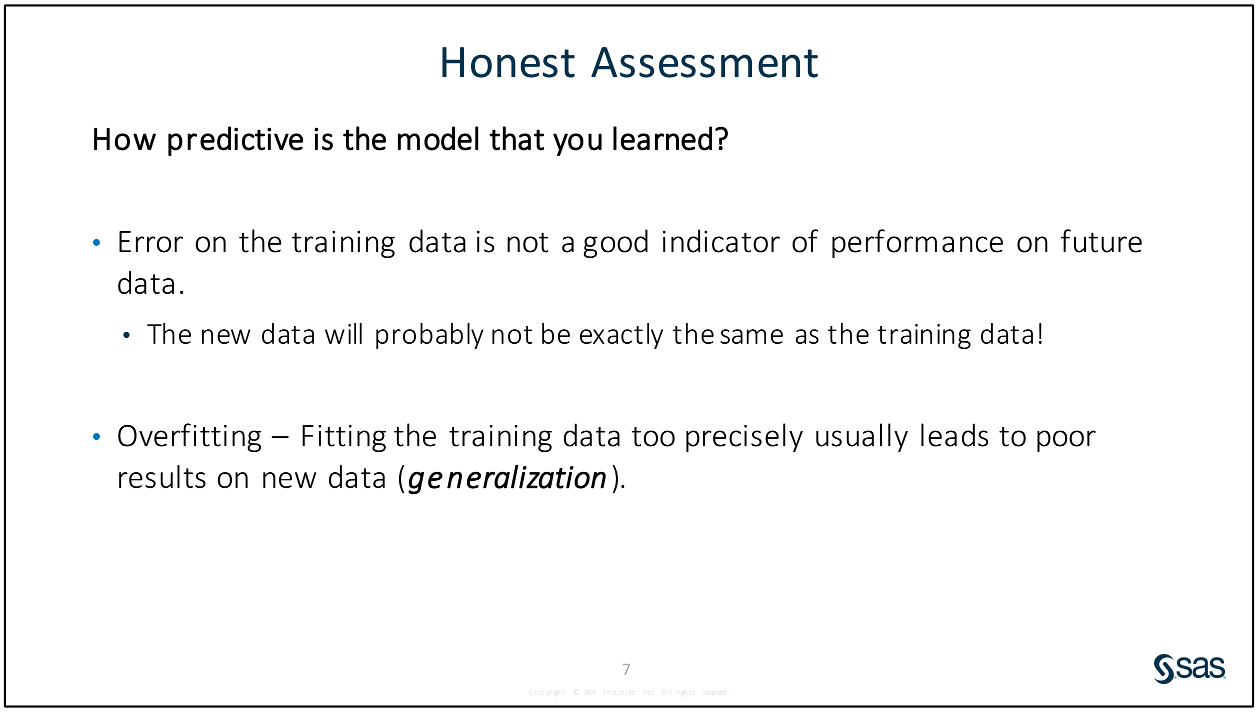
Evaluation of Model Performance

- No model is uniformly the best (particularly over time).
- Dimensions for comparison:
  - speed of training
  - speed of scoring
  - feasibility of deployment
  - noise tolerance
  - explanation ability
- Best results can come from hybrid, integrated models.

6

No model is uniformly the best, particularly when considering the deployment over time, when data changes. All models are based somehow on the data provided. The data describes the problem or the business scenario analyzed. When the scenario changes, the data change and the model can degrade in terms of predictions.

It is also important to evaluate the model according to the business needs. What is more important to a problem? The ability to explain the prediction, the model's accuracy, the speed to score, or the speed to train? In well-regulated industries, you need to explain the prediction, so some techniques are more suitable. In some business scenarios, the target might change dynamically, so the models need to be trained very fast. In some cases, the model needs to be scored in real time, so the scoring process is the most important variable in this equation. There is no universal best model. It depends on what is required in terms of problem solving and business needs.



Honest Assessment

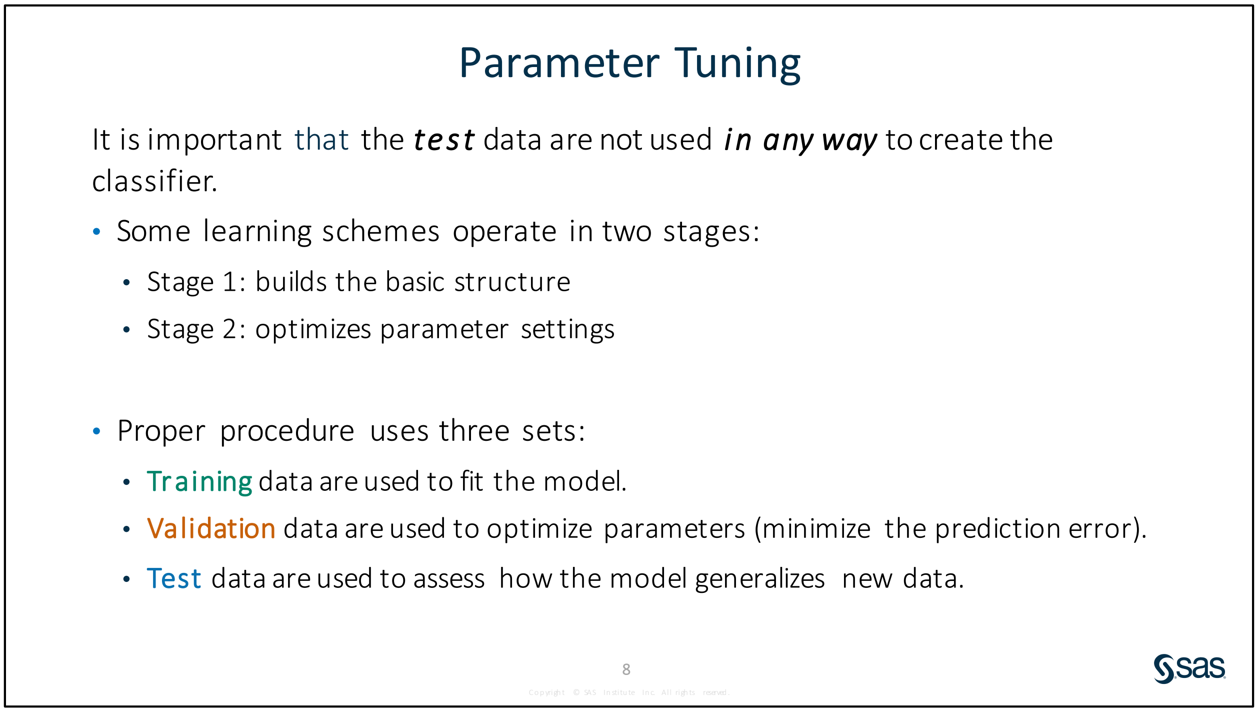
How predictive is the model that you learned?

- Error on the training data is not a good indicator of performance on future data.
  - The new data will probably not be exactly the same as the training data!
- Overfitting – Fitting the training data too precisely usually leads to poor results on new data (***generalization***).

7

The purpose of predictive modeling is generalization, which is the performance of the predictions on new data. As was stated before, evaluating the model on the same data the model was fit on usually leads to an optimistically biased assessment. The simplest strategy for correcting the optimism bias is data splitting, where a portion of the data is used to fit the model and the rest is held out for empirical validation.

The training data set is used to fit the model to the data provided. This data set enables the model to learn the relationship between the input variables (descriptors) and the target. However, overtraining the model based on this data set might lead the model to perform poorly in the new data. The model is fitted to capture the relationship between the inputs and the target for that data set. When the data set changes and new data are coming in, the model tends to fail.



Parameter Tuning

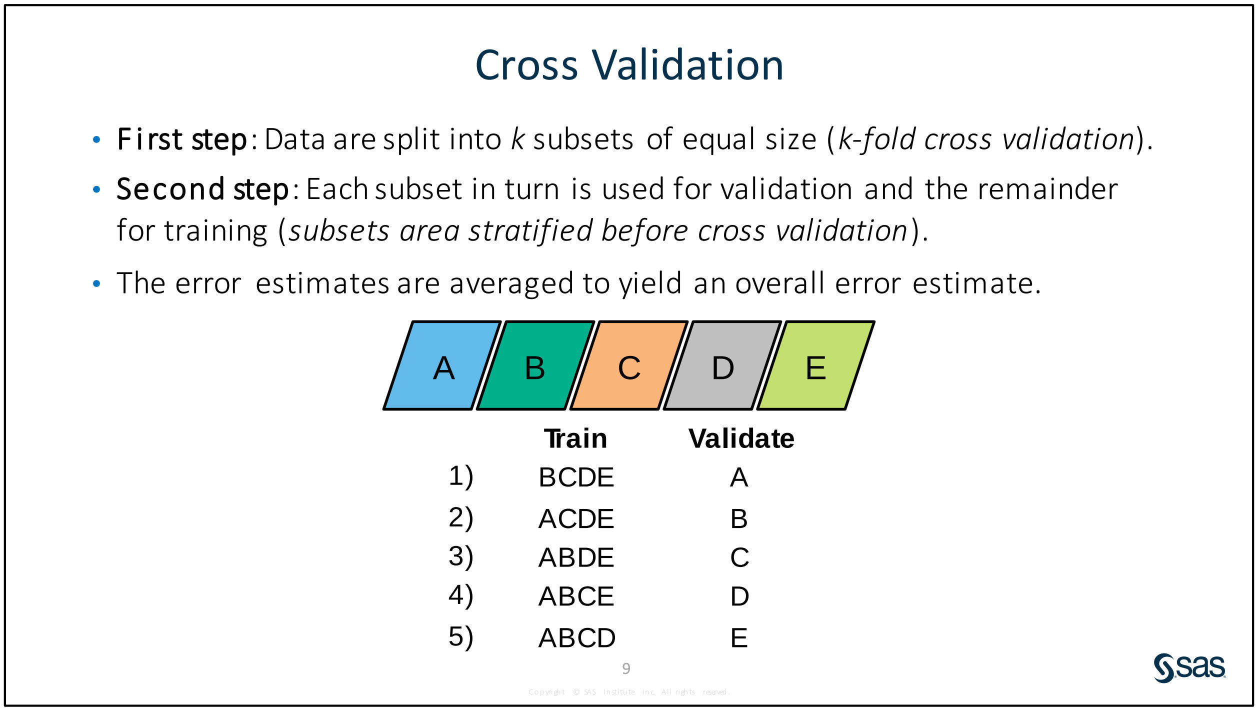
It is important that the ***test*** data are not used ***in any way*** to create the classifier.

- Some learning schemes operate in two stages:
  - Stage 1: builds the basic structure
  - Stage 2: optimizes parameter settings
- Proper procedure uses three sets:
  - Training data are used to fit the model.
  - Validation data are used to optimize parameters (minimize the prediction error).
  - Test data are used to assess how the model generalizes new data.

8

It is important that the validation data set is used to optimize the model parameters. The training data set will be used to fit the model. The test data set is used to evaluate how the model would perform based on new data. It is useful to assess how the model can generalize the new data, which most likely will differ from the one used to train and validate the model.

The cases in these data sets (training, validation, and test) should be distinct. No case should be assigned to more than one data set.



Cross Validation

- **First step**: Data are split into *k* subsets of equal size (*k-fold cross validation*).
- **Second step**: Each subset in turn is used for validation and the remainder for training (*subsets area stratified before cross validation*).
- The error estimates are averaged to yield an overall error estimate.

	Train	Validate
1)	BCDE	A
2)	ACDE	B
3)	ABDE	C
4)	ABCE	D
5)	ABCD	E

9

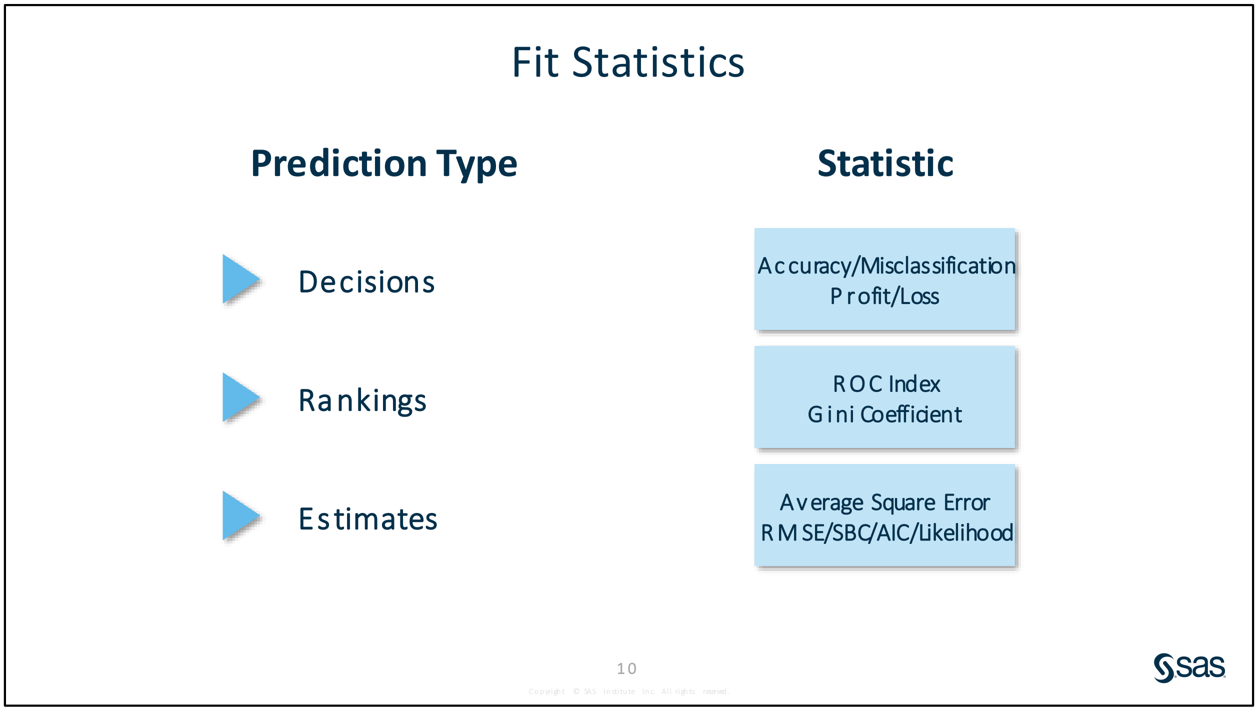
Data splitting is a simple but costly technique. When the data set is too small to split into training and validation, we can use cross validation. Cross validation avoids overlapping test sets.

- First step: Data are split into *k* subsets of equal size.
- Second step: Each subset in turn is used for validation and the remainder for training.

This is called *k-fold cross validation*.

In a k-fold cross validation, the data set is divided into k subsets. For example, in a five-fold cross validation, the initial data set is divided into A, B, C, D, and E subsets. On the first run, the subsets B, C, D, and E are used to train the model, and the subset A is used to validate the model. Then the subsets A, C, D, and E are used to train the model, and the subset B is used to validate. The process goes on until all subsets are used for training and validation.

Often the subsets are stratified before the cross validation is performed. The error estimates are averaged to yield an overall error estimate.



Fit Statistics

Prediction Type	Statistic
Decisions	Accuracy/Misclassification Profit/Loss
Rankings	ROC Index Gini Coefficient
Estimates	Average Square Error RMSE/SBC/AIC/Likelihood

10

Two factors determine the appropriate validation assessment rating, or more properly, *assessment measure*:

- the target measurement scale
- the prediction type

An appropriate measure for a binary target might not make sense for an interval target. Similarly, models tuned for decision predictions might make poor estimate predictions.

For this discussion, a binary target is assumed with a primary outcome (**target=1**) and a secondary outcome (**target=0**). The appropriate assessment measure for interval targets is noted below.

The measure that you should use to judge a model depends on the type of prediction that you want.

Consider decision predictions first. With a binary target, you typically consider two decision types:

- the primary decision, corresponding to the primary outcome
- the secondary decision, corresponding to the secondary outcome

Matching the primary decision with the primary outcome yields a correct decision called a *true positive*. Likewise, matching the secondary decision to the secondary outcome yields a correct decision called a *true negative*. Decision predictions can be rated by their *accuracy*, that is, the proportion of agreement between prediction and outcome.

Mismatching the secondary decision with the primary outcome yields an incorrect decision called a *false negative*. Likewise, mismatching the primary decision to the secondary outcome yields an incorrect decision called a *false positive*. A decision prediction can be rated by its *misclassification*--that is, the proportion of disagreement between the prediction and the outcome.

Consider ranking predictions for binary targets. With ranking predictions, a score is assigned to each case. The basic idea is to rank the cases based on their likelihood of being a primary or secondary outcome. Likely primary outcomes receive high scores and likely secondary outcomes receive low scores.

When a pair of primary and secondary cases is correctly ordered, the pair is said to be in *concordance*. Ranking predictions can be rated by their degree of concordance--that is, the proportion of such pairs whose scores are correctly ordered.

When a pair of primary and secondary cases is incorrectly ordered, the pair is said to be in *discordance*. Ranking predictions can be rated by their degree of discordance--that is, the proportion of such pairs whose scores are incorrectly ordered.

When a pair of primary and secondary cases ordered equal, the pair is said to be a *tied pair*. This implies that your model is not able to differentiate between primary and secondary outcomes. Lesser number of tied pairs is better.

Finally, consider estimate predictions. For a binary target, *estimate predictions* are the probability of the primary outcome for each case. Primary outcome cases should have a high predicted probability. Secondary outcome cases should have a low predicted probability.

The squared difference between a target and an estimate is called the *squared error*. Averaged over all cases, squared error is a fundamental assessment measure of model performance.

When calculated in an unbiased fashion, the average square error is related to the amount of bias in a predictive model. A model with a lower average square error is less biased than a model with a higher average square error.

In summary, decisions require high accuracy or low misclassification, rankings require high concordance or low discordance, and estimates require low (average) squared error.

Model fit statistics can be grouped by prediction type.

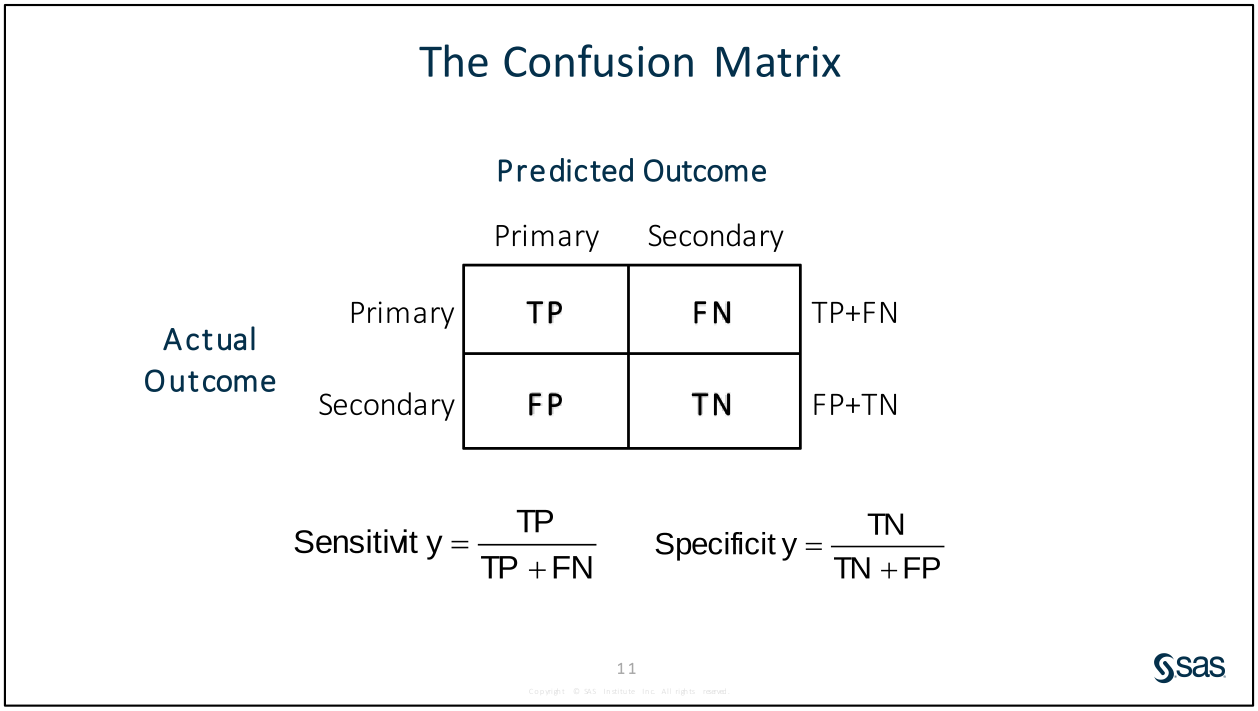
For decision prediction, the Model Comparison tool rates model performance based on accuracy or misclassification, profit or loss, and by the Kolmogorov-Smirnov (KS) statistic. Accuracy and misclassification tally the correct or incorrect prediction decisions. The Kolmogorov-Smirnov statistic describes the ability of the model to separate the primary and secondary outcomes.

Note: The Kolmogorov-Smirnov (Youden) statistic is a goodness of fit statistic that represents the maximum distance between the model ROC curve and the baseline ROC curve.

For ranking predictions, there are two closely related measures of model fit that are commonly used. The ROC index is like concordance (described earlier). The Gini coefficient (for binary prediction) equals 2 × (ROC Index – 0.5).

Note: The ROC index equals the percent of concordant cases plus one-half times the percent of tied cases.

For estimate predictions, there are at least two commonly used performance statistics. The Schwarz's Bayesian criterion (SBC) is a penalized likelihood statistic. This likelihood statistic can be thought of as a weighted average square error.



The Confusion Matrix

Predicted Outcome

Actual Outcome		Primary	Secondary	
	Primary	**TP**	**FN**	TP+FN
	Secondary	**FP**	**TN**	FP+TN

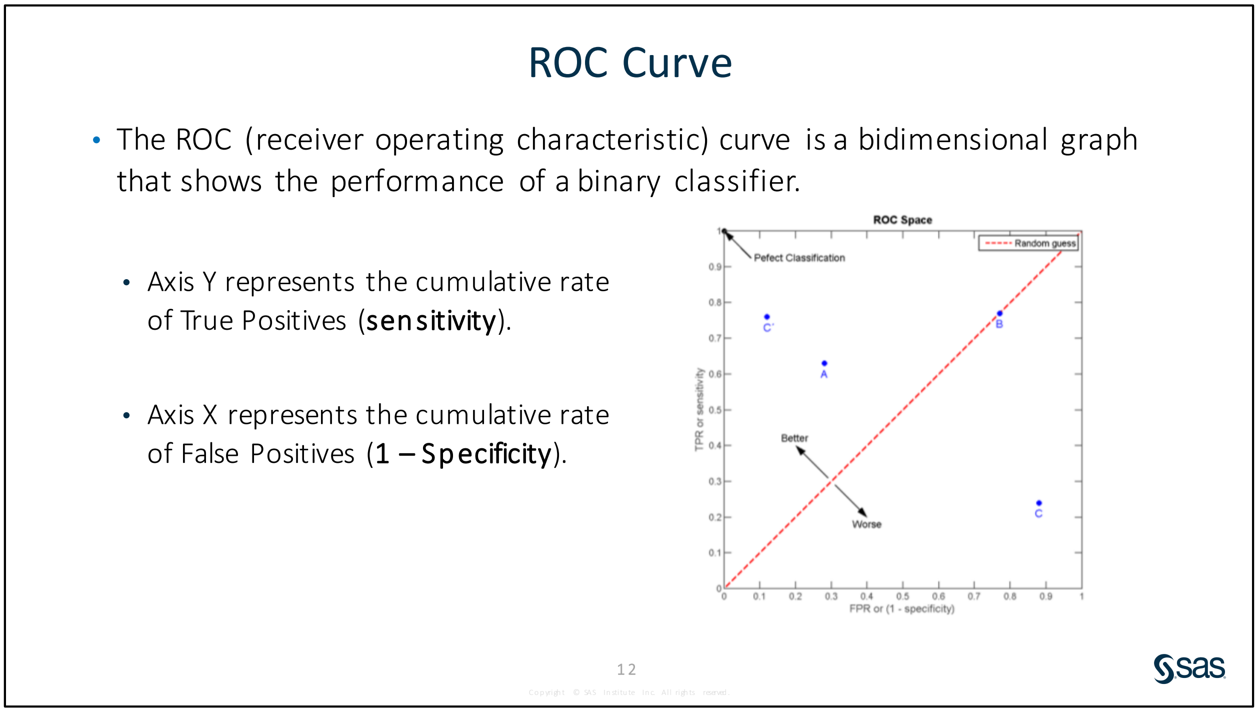
$$\text{Sensitivity} = \frac{TP}{TP + FN} \qquad \text{Specificity} = \frac{TN}{TN + FP}$$

11

The confusion matrix is a cross tabulation of the actual and predicted classes. The counts need to be adjusted if the class proportion in the training sample is not the same as in the population.

It is common practice to try to balance the classes, particularly with rare outcomes.

A confusion matrix presupposes an allocation (decision) rule. A primitive rule allocates cases to the class with the greatest posterior probability. For binary targets, this corresponds to a 50% cutoff on the posterior probability.



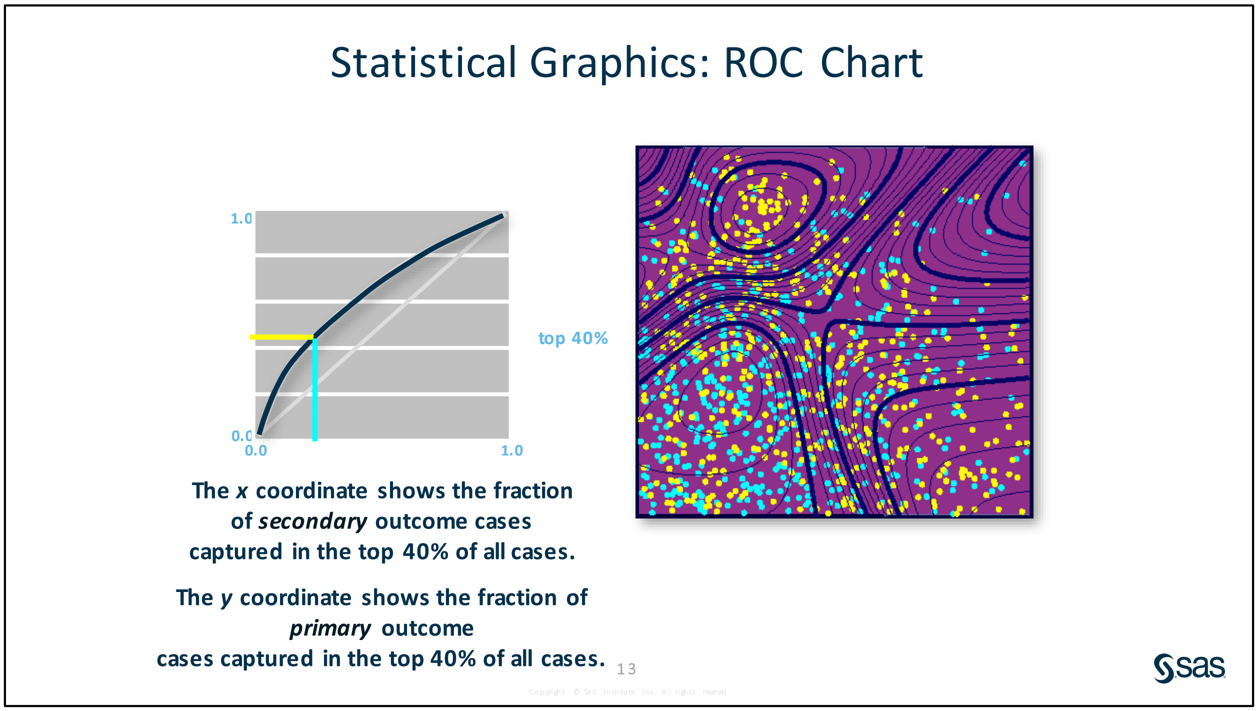
ROC Curve

- The ROC (receiver operating characteristic) curve is a bidimensional graph that shows the performance of a binary classifier.
  - Axis Y represents the cumulative rate of True Positives (**sensitivity**).
  - Axis X represents the cumulative rate of False Positives (**1 – Specificity**).

In addition to numeric measures of model performance, data scientists often use graphical tools to assess models as well. One commonly used graphical measure of assessment is the ROC chart. To understand ROC chart, two important measures are worth discussing: sensitivity and specificity.

The *sensitivity* of a model is the true positive rate.

The *specificity* of a model is the true negative rate.



Statistical Graphics: ROC Chart

The *x* coordinate shows the fraction of *secondary* outcome cases captured in the top 40% of all cases.

The *y* coordinate shows the fraction of *primary* outcome cases captured in the top 40% of all cases.

- Axis Y represents the cumulative rate of true positives (**sensitivity**).
- Axis X represents the cumulative rate of false positives (**1 – specificity**).

A ROC curve shows the probability distribution just for the events, the positive class. It compares the events predicted correctly (***true positives***) against the events predicted incorrectly (***false positives***). The higher the sensitivity (true positive rate close to 1) and the lower the false negative rate (specificity close to 1 – one minus specificity close to zero), the better the model. The ROC index gives us the area under the ROC curve: the greater the area, the higher the index.

Therefore, the ROC chart plots how the true positive rate changes as the false positive rate changes. The classification accuracy of a model is demonstrated by the degree that the ROC curve pushes upward and to the left.

Interpreting ROC charts can be easier when it is understood how they are constructed. We discuss their construction over the next several slides.

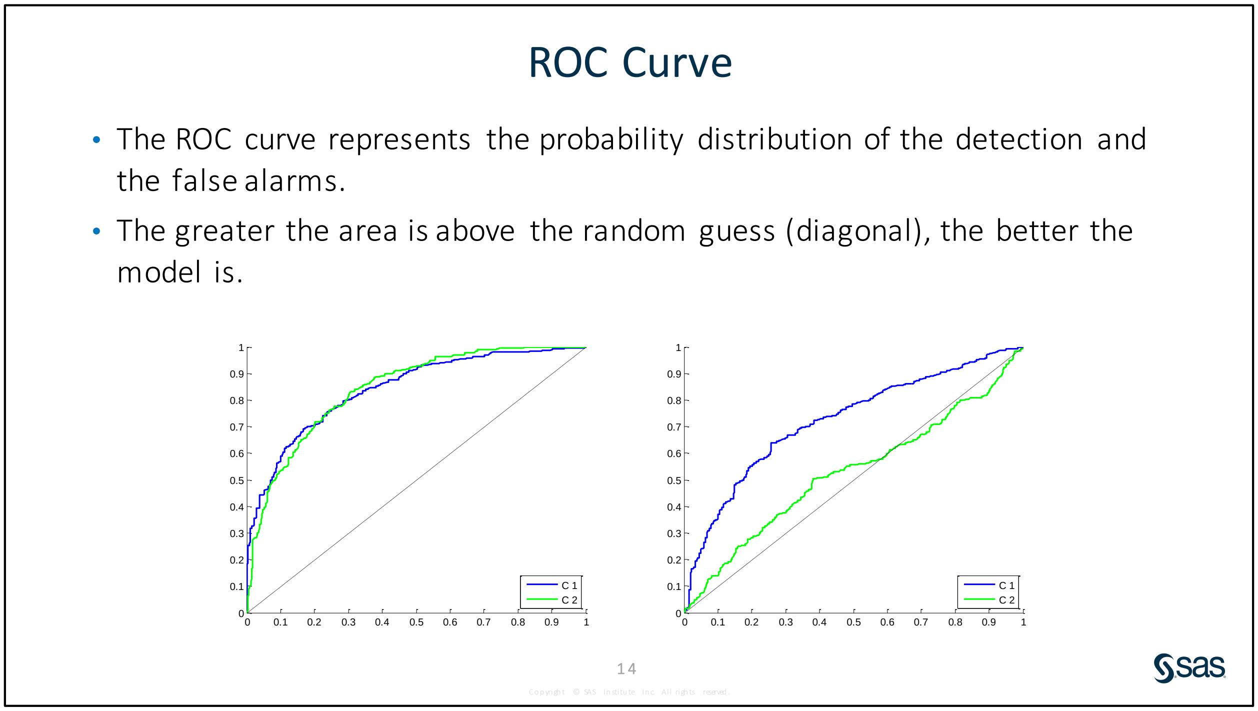
To create a ROC chart, predictions are generated for a set of validation data. For chart generation, the predictions must be rankings or estimates. The validation data are sorted from high to low (either scores or estimates). Each point on the ROC chart corresponds to a specific fraction of the sorted data.

For example, the red point on the ROC chart corresponds to the selection of 40% of the validation data with the highest predicted probabilities.

The vertical, or *y*, coordinate of the red point indicates the fraction of primary outcome cases captured in the gray region (here, approximately 45%).

The horizontal, or *x*, coordinate of the red point indicates the fraction of secondary outcome cases captured in the gray region (here, approximately 25%).

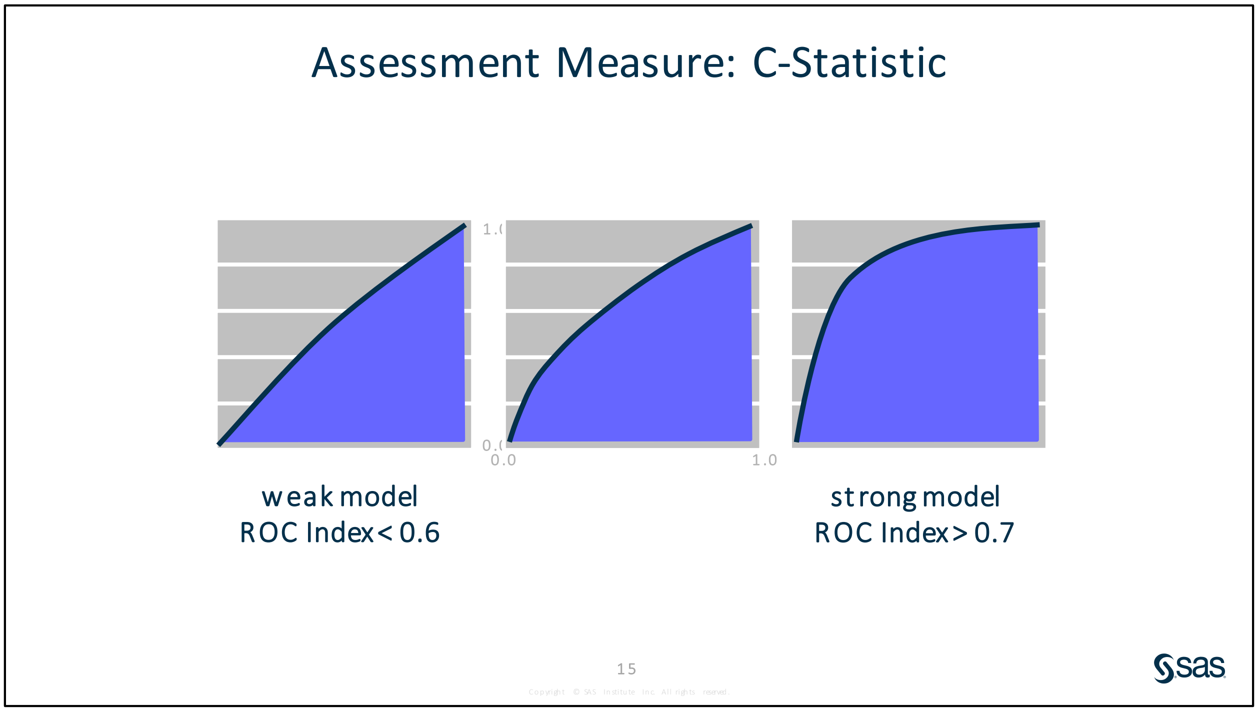
The ROC chart represents the union of similar calculations for all selection fractions.



ROC Curve

- The ROC curve represents the probability distribution of the detection and the false alarms.
- The greater the area is above the random guess (diagonal), the better the model is.

The ROC (receiver operating characteristic) curve is a bi-dimensional graph that shows the performance of a binary classifier. ROC chart displays the ability of a model to avoid false positive and false negative classifications. A false positive classification means that an observation has been identified as an event when it is a nonevent. A false negative classification means that an observation has been identified as a nonevent when it is an event.



Assessment Measure: C-Statistic

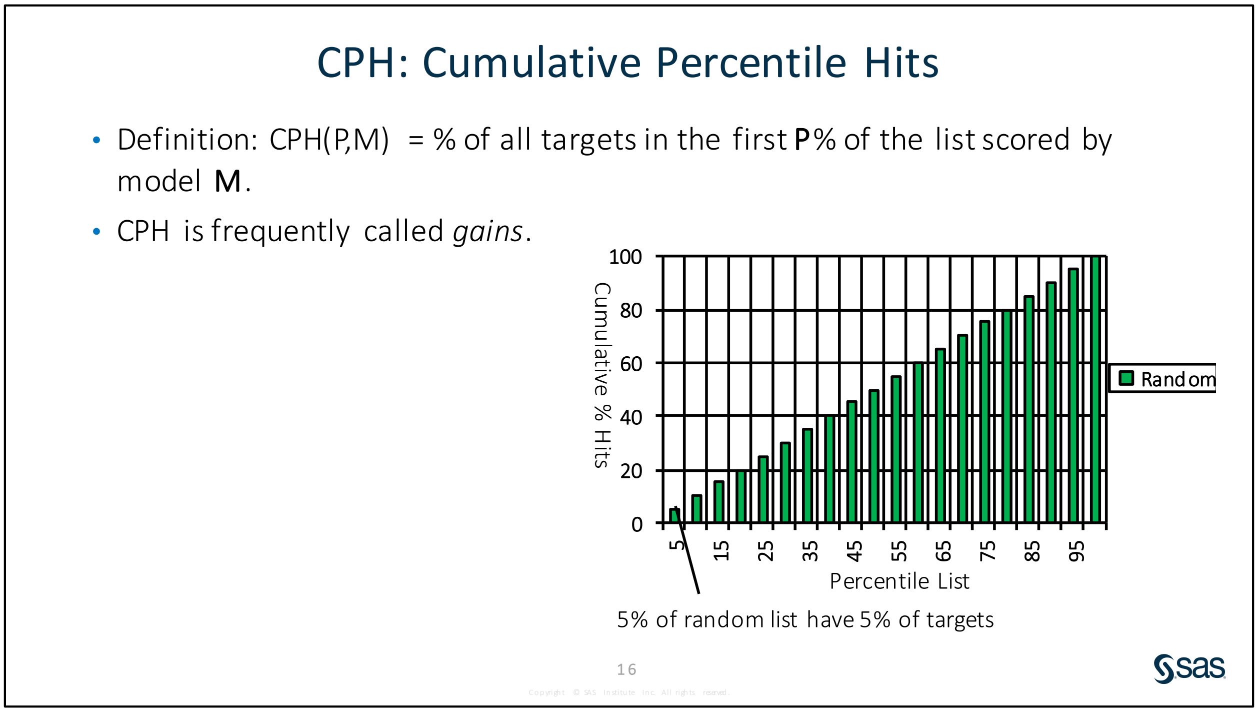
weak model
ROC Index < 0.6

strong model
ROC Index > 0.7

15

The ROC chart provides a nearly universal diagnostic for predictive models. Models that capture primary and secondary outcome cases in a proportion approximately equal to the selection fraction are weak models (left). Models that capture mostly primary outcome cases without capturing secondary outcome cases are strong models (right).

The tradeoff between primary and secondary case capture can be summarized by the area under the ROC curve. This area can be referred to as the C-Statistic. (In machine learning literature, it is more commonly called the ROC Index.) Perhaps surprisingly, the C-Statistic is closely related to concordance, the measure of correct case ordering.



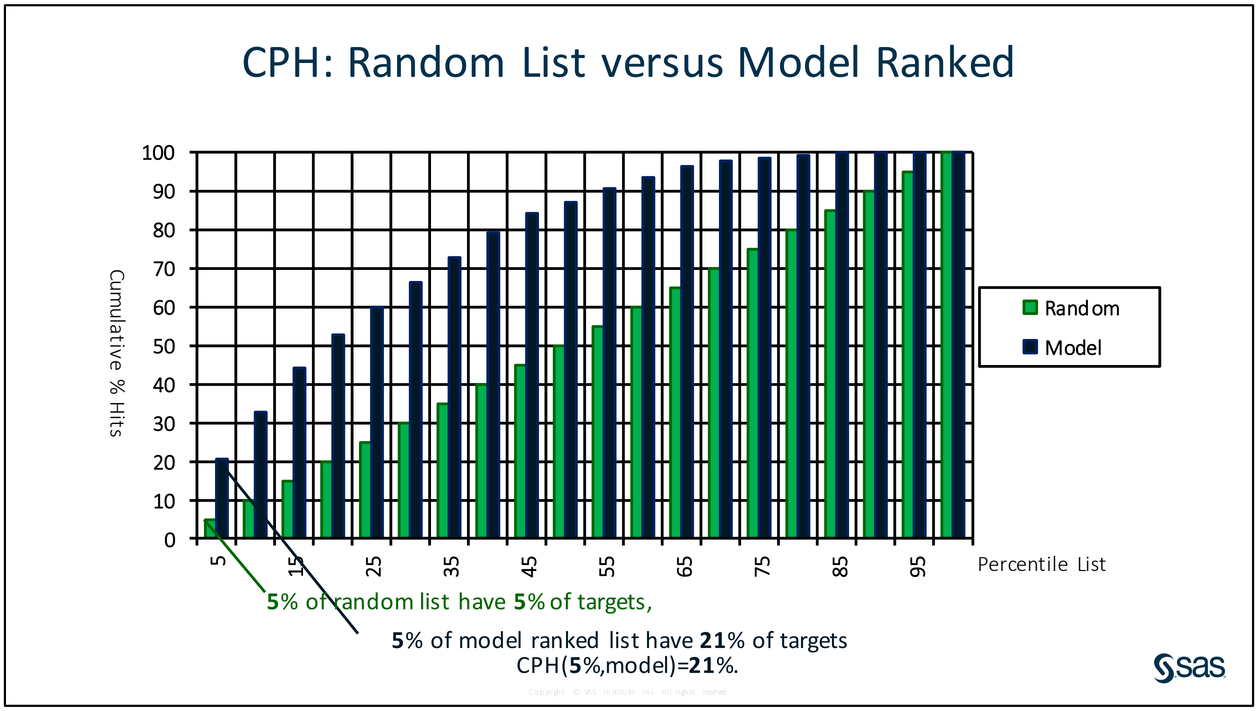
CPH: Cumulative Percentile Hits

- Definition: CPH(P,M) = % of all targets in the first **P**% of the list scored by model **M**.
- CPH is frequently called *gains*.

5% of random list have 5% of targets

16

Cumulative percentile hits and lift charts are a graphical representation of the advantage of using a predictive model to choose which customers to contact in comparison to no model. The lift chart shows how much more likely we are to receive respondents than if we contact a random sample of customers.

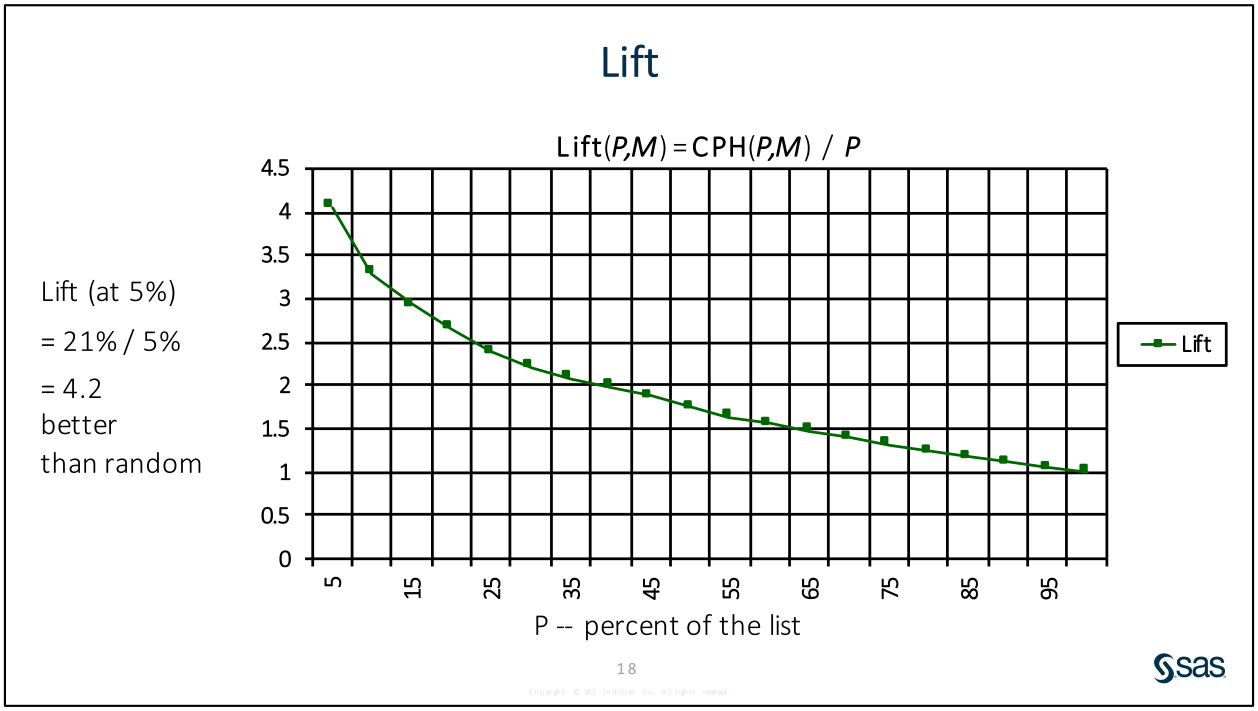


CPH: Random List versus Model Ranked

5% of random list have **5**% of targets,

5% of model ranked list have **21**% of targets
CPH(**5**%,model)=**21**%.

The lift chart uses the predictions of the response model to calculate the percentage of positive responses according to the percent of customers contacted and maps these points to create the lift chart. It calculates the points on the lift curve by determining the ratio between the result predicted by the predictive model and the result using no model.



Lift

Lift (at 5%)

= 21% / 5%

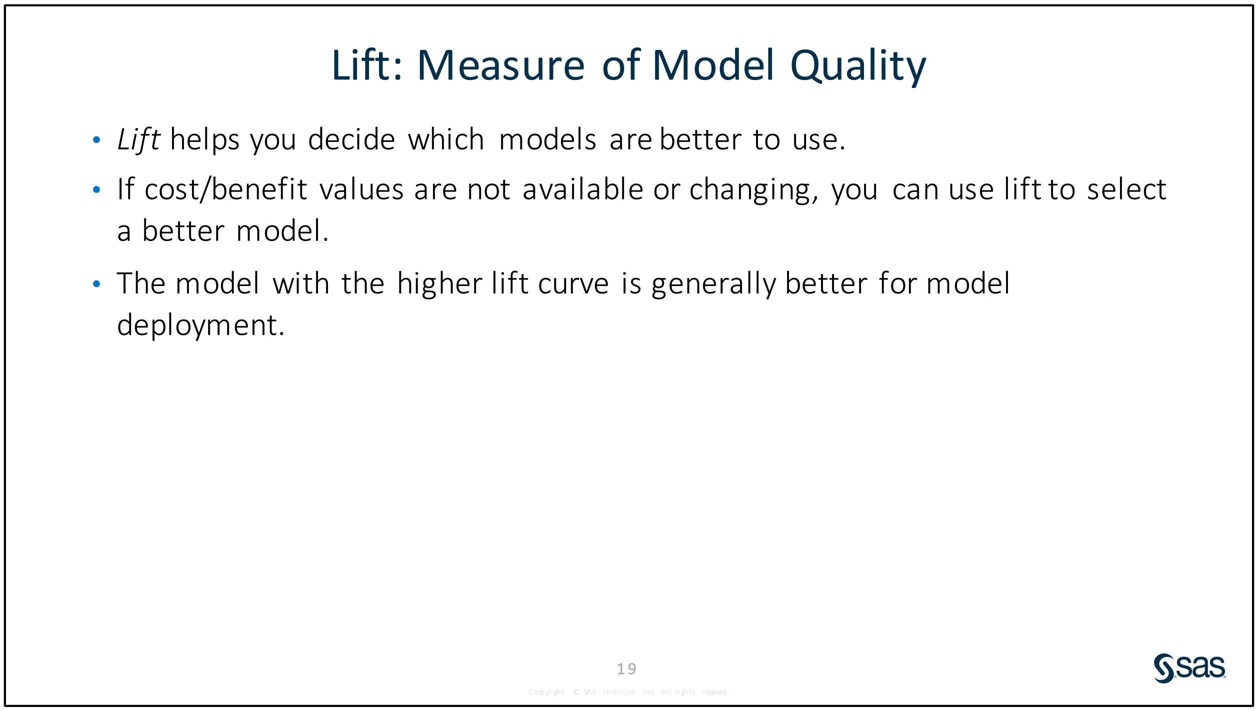
= 4.2

better

than random

18

For example, if you contact 5% of all customers, using no model, you should get 5% of responders. By using the predictive model, you should get 21% of responders contacting the same 5% of customers. The *y* value of the lift curve at 5% is 21/5=4.2.



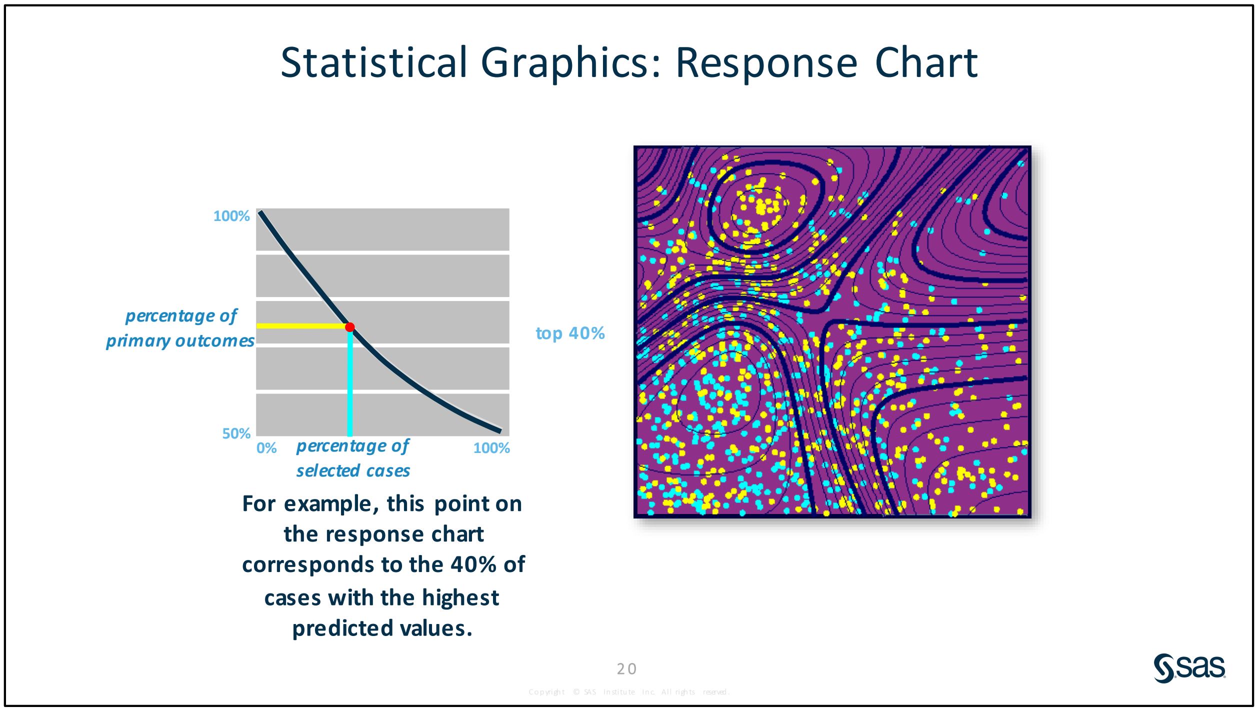
Lift: Measure of Model Quality

- *Lift* helps you decide which models are better to use.
- If cost/benefit values are not available or changing, you can use lift to select a better model.
- The model with the higher lift curve is generally better for model deployment.

19

The Lift chart aims to select the best model in a particular business scenario (for example, considering a limit in the budget or an operational capacity to contact a fixed number of prospects or customers.

For example, the Lift charts would help you decide which model you should choose to deploy according to the gain achieved in different percentiles for the population. If you expect to deploy a campaign to contact 20% of your customers, you should choose the model that performs best on this percentile.



Statistical Graphics: Response Chart

For example, this point on the response chart corresponds to the 40% of cases with the highest predicted values.

20

Because lift is based on response rate, it might be helpful to understand how a response chart is constructed.

As with ROC charts, a model is applied to validation data to sort the cases from highest to lowest (again, by prediction rankings or estimates). Each point on the response chart corresponds to a selected fraction of cases.

For example, the red point on the response chart corresponds to the selection of 40% of the validation data with the highest predicted probabilities.

The *x* coordinate of the red point is simply the selection fraction (in this case, 40%). This is illustrated with the vertical blue line in the plot above.

The vertical coordinate for a point on the response chart is the proportion of primary outcome cases in the selected fraction. This is illustrated with the horizontal yellow line in the plot above. It is often called the *cumulative percent response* but is more widely known as *cumulative gain* in the predictive modeling literature. Dividing cumulative gain by the primary outcome proportion yields *lift*.

Plotting cumulative gain for all selection fractions yields a *gains chart* (also called a *response chart*). Notice that when all cases are selected, the cumulative gain equals the overall primary outcome proportion.

Essential Deployment Tasks

Deployment
Data
Discovery

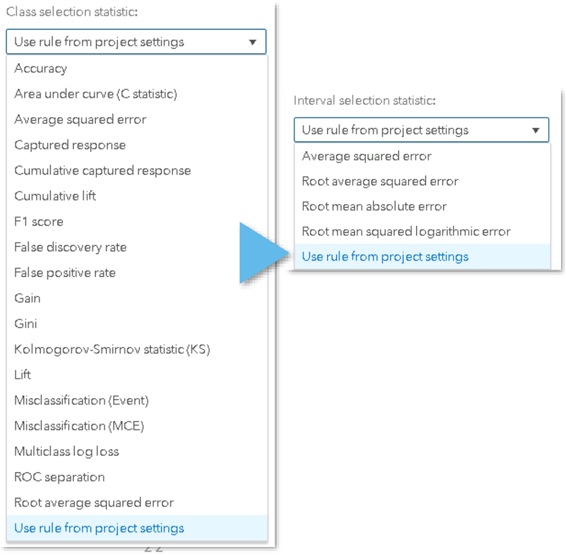
- Assess models.
- Compare models.
- Score the champion model.
- Monitor model performance over time.
- Update the model as needed.

21

Comparing several models is essential to determine the champion model that is scored last.

Comparing Models in Model Studio

- Model Comparison Node
  - Compares models within a pipeline
- Pipeline Comparison Tab
  - Compares models across pipelines


Class selection statistic:
Use rule from project settings
Accuracy
Area under curve (C statistic)
Average squared error
Captured response
Cumulative captured response
Cumulative lift
F1 score
False discovery rate
False positive rate
Gain
Gini
Kolmogorov-Smirnov statistic (KS)
Lift
Misclassification (Event)
Misclassification (MCE)
Multiclass log loss
ROC separation
Root average squared error
Use rule from project settings
Interval selection statistic:
Use rule from project settings
Average squared error
Root average squared error
Root mean absolute error
Root mean squared logarithmic error
Use rule from project settings

22

In Model Studio, you can compare models within a pipeline as well as across several pipelines using a Model Comparison node and a Pipeline Comparison tab.

The Model Comparison node is automatically added to a pipeline when a Supervised Learning node is also added. The Model Comparison node enables you to compare the performance of competing models using various benchmarking criteria. There are many assessment criteria that can be used to compare models. For class targets, these include measures of error, lift-based measures, and measures derived from the ROC curve. You can select the measure and specify the depth to use when applying a lift-based measure or the cutoff to use when applying an ROC-based measure. For interval targets, there are various measures of error available for choosing the champion model. All measures of assessment are computed for each of the data partitions that are available (train, validate, and test). You can also select which data partition to use for selecting the champion.

The Pipeline Comparison tab compares only the champion models for each pipeline. The selected model from the Model Comparison node of each pipeline is added to the Pipeline Comparison tab. This enables you to compare models from the different pipelines in your project and to select a champion model. To add models that were not selected by the Model Comparison node to the Pipeline Comparison tab, right-click the given model, and select **Add challenger model**.

Model Studio gives you several options of model assessment and comparison. Below is a glossary of assessment measures offered.

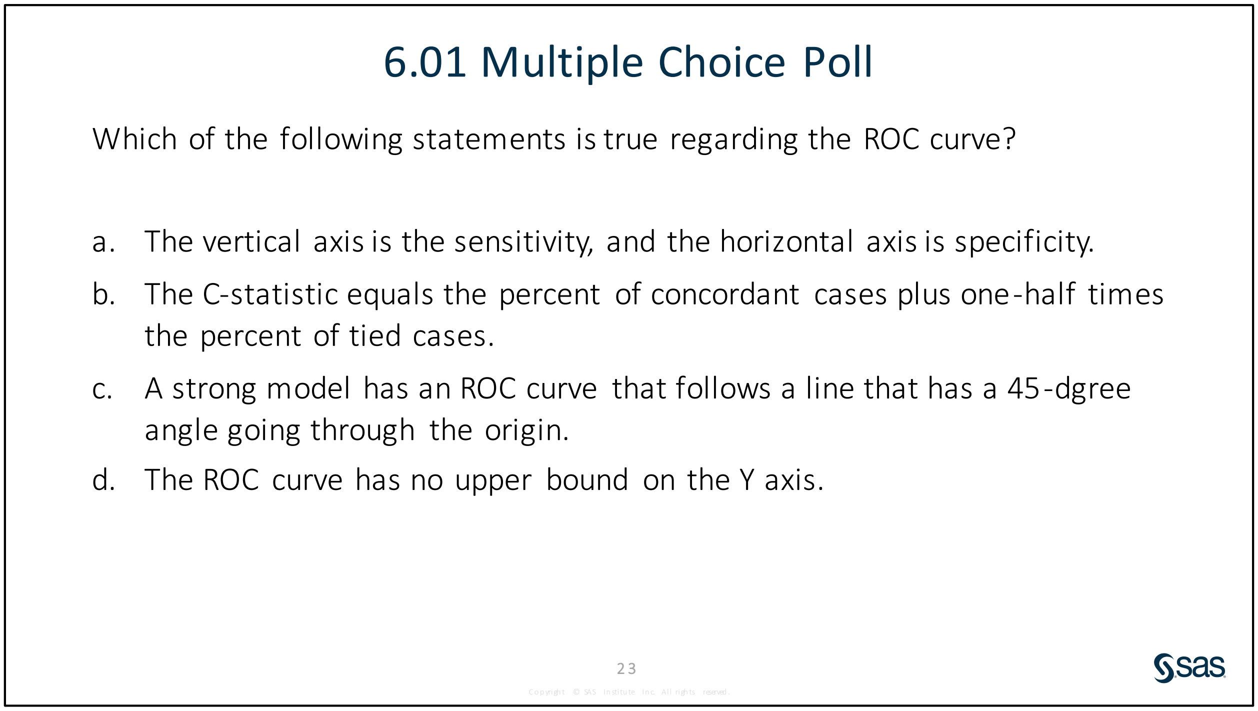
<u>Class selection statistics:</u>

Accuracy	A measure of how many observations are correctly classified for each value of the response variable. It is the number of event and non-event cases classified correctly, divided by all cases.
Area under the curve (C statistic)	It is a measure of goodness of fit for binary outcome. It is the concordance rate and it is calculated as the area under the curve.
Average squared error	The sum of squared errors (SSE) divided by the number of observations.
Captured response	The number of events in each bin divided by the total number of events.
Cumulative captured response	Cumulative the captured response.
Cumulative lift	Cumulative lift up to and including the specified percentile bin of the data, sorted in descending order of the predicted event probabilities.
F1 score	The weighted average of precision (positive predicted value) and recall (sensitivity). It is also known as the F-score or F-measure.
False discovery rate	The expected proportion of type error I – incorrectly reject the null hypothesis (false positive rate).
False positive rate	The number of positive cases misclassified (as negative).

Gain	Similar to a lift chart. It equals the expected response rate using the predictive model divided by the expected response rate from using no model at all.
Gini	A measure of the quality of the model. It has values between -1 and 1. Closer to 1 is better. It is also known as Somer's D.
Kolmogorov-Smirnov statistic (KS)	A goodness-of-fit statistic that represents the maximum separation between the model ROC curve and the baseline ROC curve.
KS (Youden)	A goodness-of-fit index that represents the maximum separation between the model ROC curve and the baseline ROC curve.
Lift	A measure of the advantage (or lift) of using a predictive model to improve on the target response versus not using a model. It is a measure of the effectiveness of a predictive model calculated as the ratio between the results obtained with and without the predictive model. The higher the lift in the lower percentiles of the chart, the better the model is.
Misclassification (Event)	Considers only the classification of the event level versus all other levels. Thus, a non-event level classified as another non-event level does not count in the misclassification. For binary targets, these two measures are the same. It is computed in the context of the ROC report. That is, at each cutoff value, this measure is calculated.
Misclassification (MCE)	A measure of how many observations are incorrectly classified for each value of the response variable. This is the true misclassification rate. That is, every observation where the observed target level is predicted to be a different level counts in the misclassification rate.
Multiclass log loss	The loss function applied to multinomial target. It is the negative log-likelihood of the true labels given a probabilistic classifier's prediction.
ROC separation	The area under the ROC curve is the accuracy. The ROC separation enables you to change the ROC-based cutoff and evaluate the model's performance under different ranges of accuracy.
Root average squared error	It is the square root of the average differences between the prediction and the actual observation.

Interval selection statistics:

Average squared error	The sum of squared errors (SSE) divided by the number of observations.
Root average squared error	It is the square root of the average squared differences between the prediction and the actual observation.
Root mean absolute error	It is the square root of the average differences between the prediction and the actual observation, not considering the direction of the error.
Root mean squared logarithmic error	It is the square root of the average squared differences between the prediction and the actual observation. The differences between the prediction and actual observation is measure by the log function.



6.01 Multiple Choice Poll

Which of the following statements is true regarding the ROC curve?

a. The vertical axis is the sensitivity, and the horizontal axis is specificity.

b. The C-statistic equals the percent of concordant cases plus one-half times the percent of tied cases.

c. A strong model has an ROC curve that follows a line that has a 45-dgree angle going through the origin.

d. The ROC curve has no upper bound on the Y axis.

Comparing Multiple Models of a Single Pipeline in Model Studio

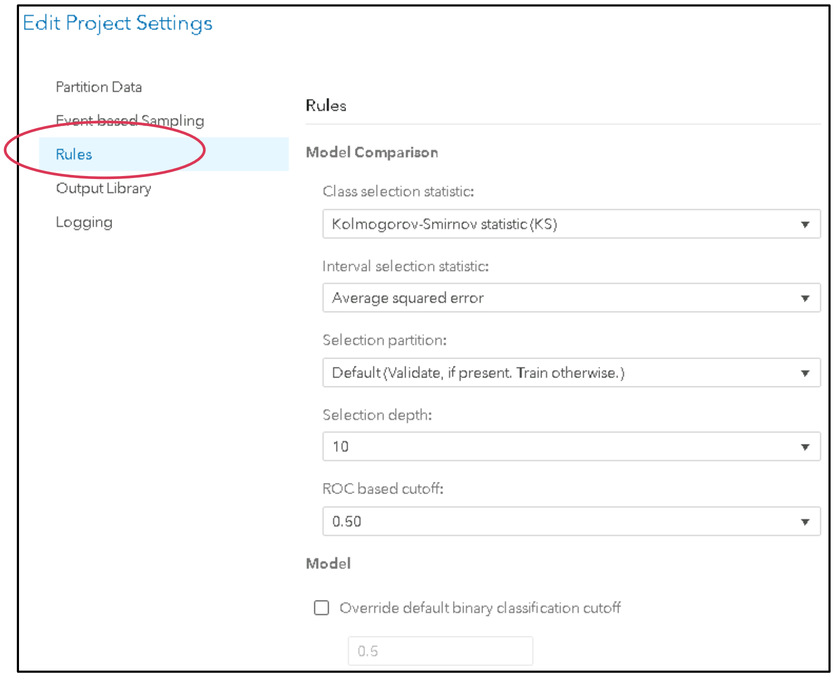
In this demonstration, you run the Model Comparison node in the Chapter 3 pipeline. You compare the models' performances based on different fit statistics.

Note: Although not shown here, you could look at the results of the Model Comparison node from any of the other pipelines where models were built.

The Model Comparison node enables you to compare the performance of competing models using various assessment measures. There are many criteria that can be used to compare models. For class targets, there are 18 different measures, including measures of error, lift, and ROC. You can select the measure and specify the depth to use when applying a lift-based measure or the cutoff to use when applying an ROC-based measure. For interval targets, there are four measures of error: ASE, RASE, RMAE, and RMSLE. All measures of assessment are computed for each of the data partitions that are available (train, validate, and test). You can also select which data partition to use for selecting the champion.

Note: If multiple supervised learning nodes are connected to the Model Comparison node, then only successfully completed models are compared. Models that have failed or been stopped are not considered. The selected model from the Model Comparison node of each pipeline is added to the Pipeline Comparison tab. This enables you to compare models from the different pipelines in your project and to select a champion model.

1. You can change the default assessment measures on the **Project settings** option under the **Rules** property. (Recall that the short-cut button for **Project settings** is found in the upper right corner of the project window.)


Edit Project Settings

Rules

Model Comparison

Class selection statistic: Kolmogorov-Smirnov statistic (KS)

Interval selection statistic: Average squared error

Selection partition: Default (Validate, if present. Train otherwise.)

Selection depth: 10

ROC based cutoff: 0.50

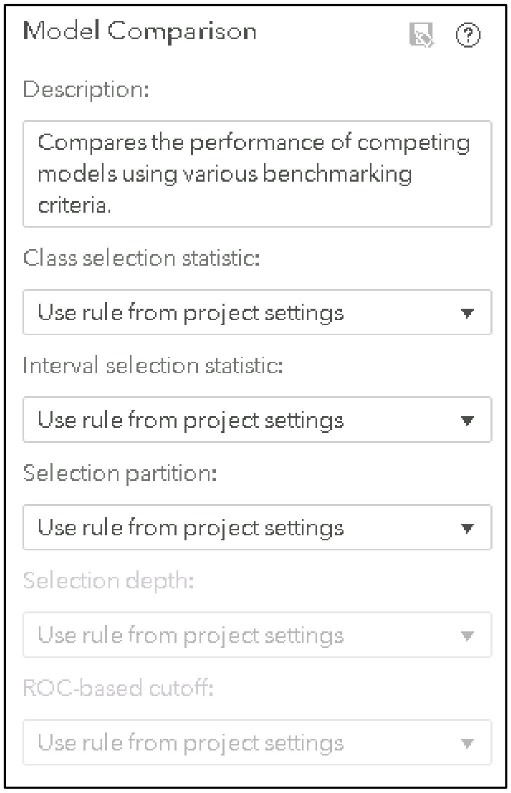
Model

☐ Override default binary classification cutoff

0.5

2. Click **Cancel** if you opened the Project settings window.

3. You can also change the default settings under the properties for the Model Comparison node. Click the **Chapter 3 pipeline** tab to open it. Select the **Model Comparison** node. Its properties are shown below.



Model Comparison

Description:

Compares the performance of competing models using various benchmarking criteria.

Class selection statistic:

Use rule from project settings

Interval selection statistic:

Use rule from project settings

Selection partition:

Use rule from project settings

Selection depth:

Use rule from project settings

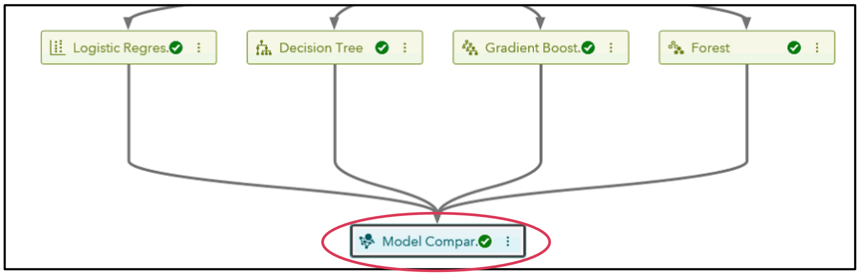
ROC-based cutoff:

Use rule from project settings

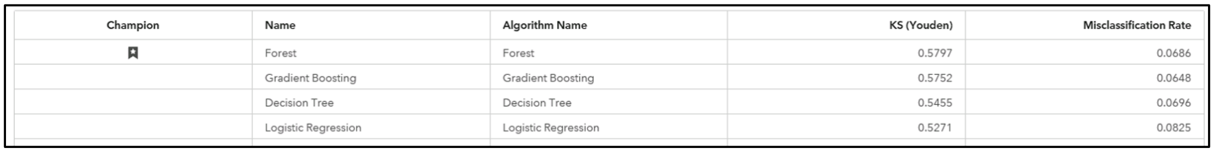
Note: The complete list of assessment measures is described in the course notes before the current demonstration.

4. It is possible that tree-based models were built and assessed individually, but the Model Comparison node was not run. Right-click the **Model Comparison** node and select **Run**.

5. When the Model Comparison node is done (the green check mark is visible), right-click the **Model Comparison** node and select **Results**.

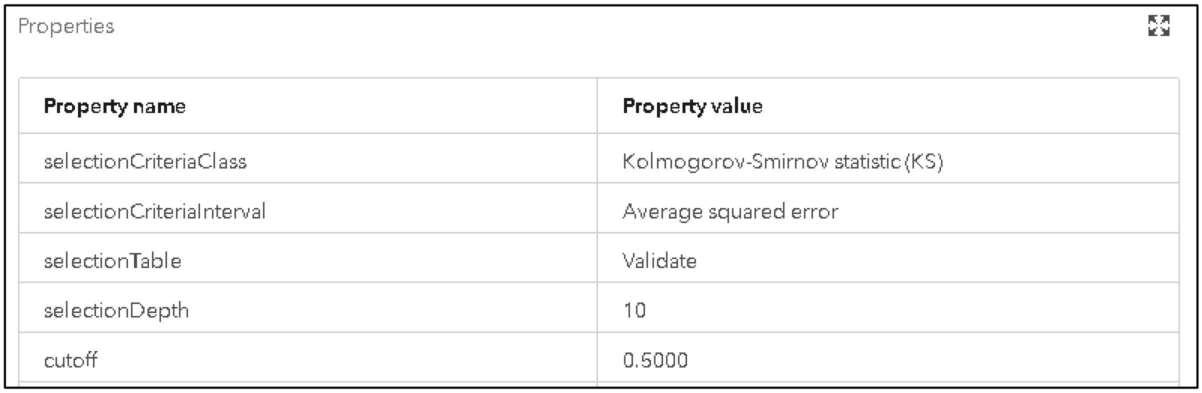


The first results table shows the champion model based on the assessment measure selected. (This comparison does not include the models developed during the exercises by using the Autotune feature).



Champion	Name	Algorithm Name	KS (Youden)	Misclassification Rate
🔖	Forest	Forest	0.5797	0.0686
	Gradient Boosting	Gradient Boosting	0.5752	0.0648
	Decision Tree	Decision Tree	0.5455	0.0696
	Logistic Regression	Logistic Regression	0.5271	0.0825

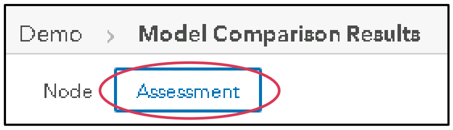
The criteria used to evaluate the models and select the champion are shown in the Properties table.



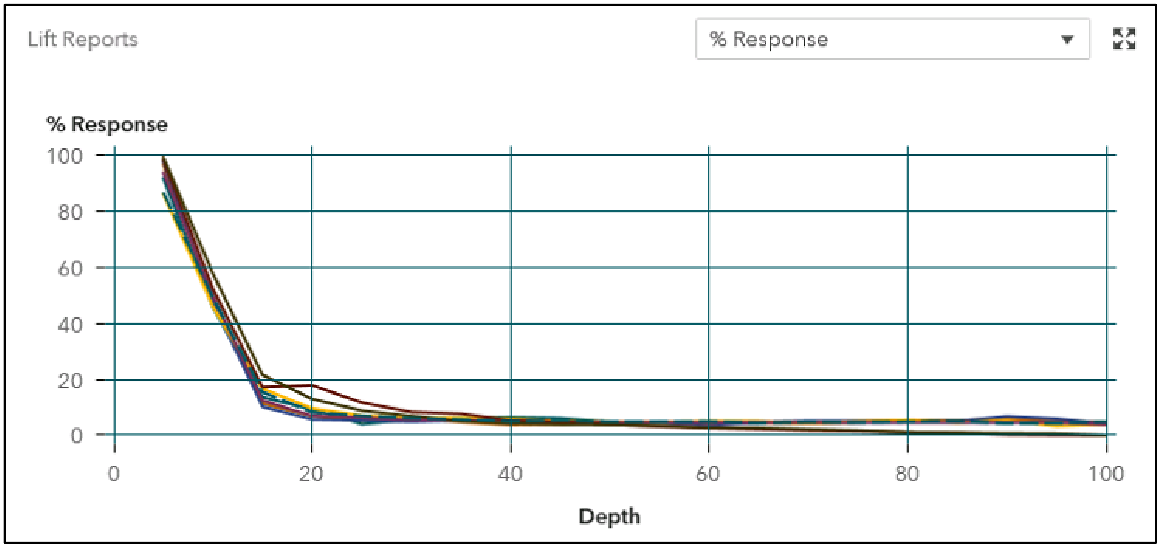
Properties

Property name	Property value
selectionCriteriaClass	Kolmogorov-Smirnov statistic (KS)
selectionCriteriaInterval	Average squared error
selectionTable	Validate
selectionDepth	10
cutoff	0.5000

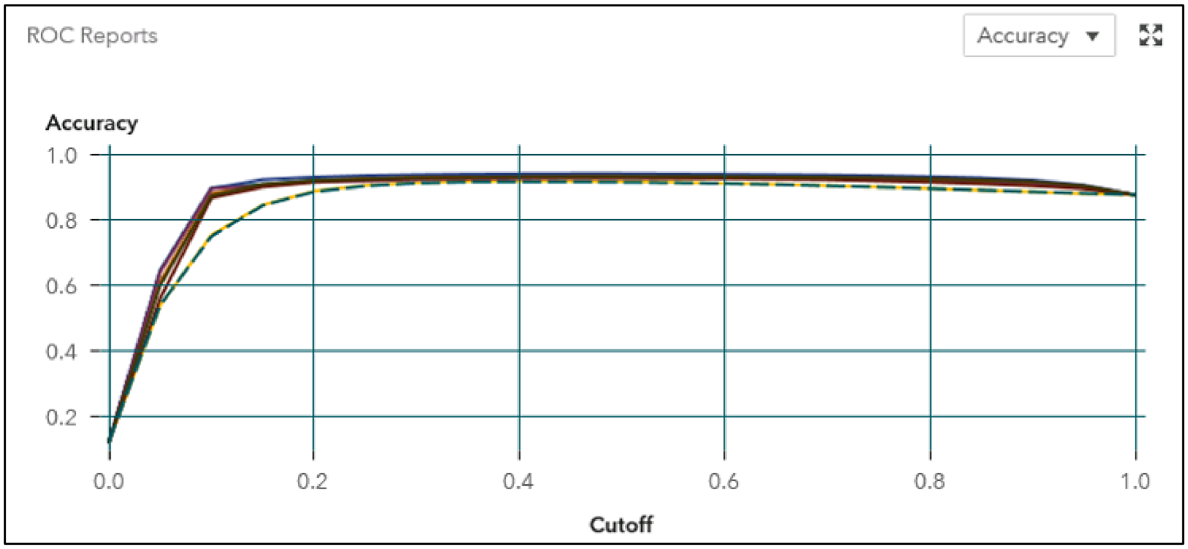
6. Click the **Assessment** tab to see more results.



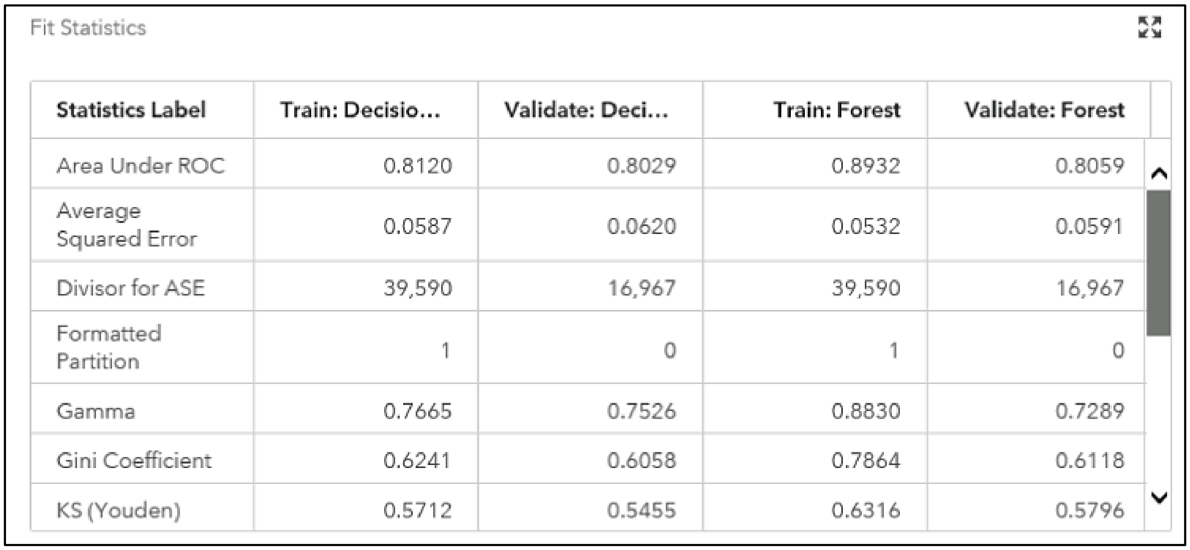
On the Assessment tab, you can find two plots and one table presenting the performance and the fit statistics for all the models compared. The first plot shows the Lift report based on **% Response**. You also have the options to see the models' performance based on the **Cumulative % Response**, **% Captured Response**, **Cumulative % Captured Response**, **Cumulative Lift**, **Gain**, and **Lift**.



The second plot shows the ROC report based on **Accuracy**. You also have the options to see the models' performances based on the F1 Score and ROC.



Finally, the Fit Statistics table shows all models' performances based on the data partitions defined in the project (train, validate, and test) for a series of fit statistics, such as Area Under ROC, Average Square Error, Gini Coefficient, and KS, among others.


Fit Statistics

Statistics Label	Train: Decisio...	Validate: Deci...	Train: Forest	Validate: Forest
Area Under ROC	0.8120	0.8029	0.8932	0.8059
Average Squared Error	0.0587	0.0620	0.0532	0.0591
Divisor for ASE	39,590	16,967	39,590	16,967
Formatted Partition	1	0	1	0
Gamma	0.7665	0.7526	0.8830	0.7289
Gini Coefficient	0.6241	0.6058	0.7864	0.6118
KS (Youden)	0.5712	0.5455	0.6316	0.5796

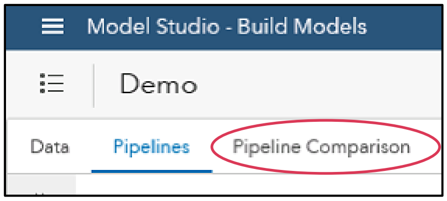
7. Close the results.

End of Demonstration

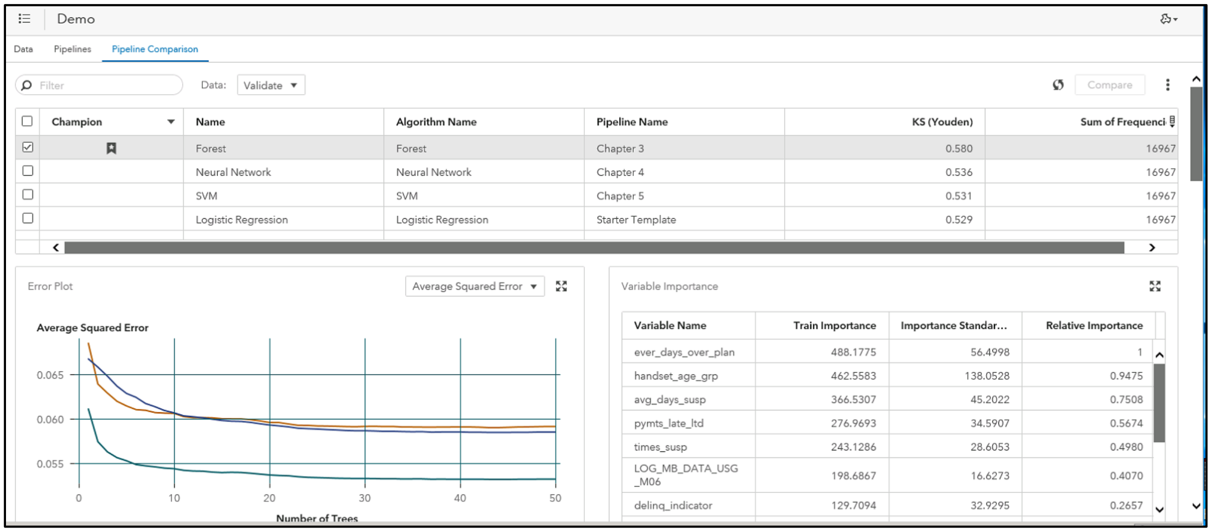
Comparing Multiple Models across Pipelines Using the Pipeline Comparison Tab and Registering the Champion

Pipeline Comparison enables you to compare the best models from each pipeline created. In addition to that, it enables you to register the champion model and use it in the Manage Models tool. (The models built in student exercises and self-study demonstrations are ignored in this demonstration.)

1. Click **Pipeline Comparison**.



At the top, you see the champion model from each pipeline as well as the model deemed the overall champion in the pipeline comparison--the champion of champions. In addition, several charts and tables are provided that summarize the performance of the overall champion model, show the variable importance list of the model, provide training and score codes, and show other outcomes from the selected best model. The default assessment measure for Pipeline Comparison is Kolmogorov-Smirnov (KS).

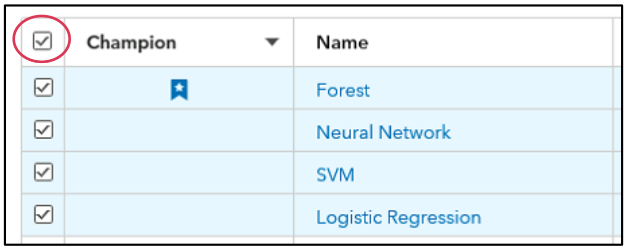


☐	Champion	Name	Algorithm Name	Pipeline Name	KS (Youden)	Sum of Frequenci
☑	🔖	Forest	Forest	Chapter 3	0.580	16967
☐		Neural Network	Neural Network	Chapter 4	0.536	16967
☐		SVM	SVM	Chapter 5	0.531	16967
☐		Logistic Regression	Logistic Regression	Starter Template	0.529	16967

Variable Name	Train Importance	Importance Standar...	Relative Importance
ever_days_over_plan	488.1775	56.4998	1
handset_age_grp	462.5583	138.0528	0.9475
avg_days_susp	366.5307	45.2022	0.7508
pymts_late_ltd	276.9693	34.5907	0.5674
times_susp	243.1286	28.6053	0.4980
LOG_MB_DATA_USG_M06	198.6867	16.6273	0.4070
delinq_indicator	129.7094	32.9295	0.2657

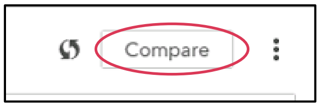
All the results shown are for the overall champion model only. There might be a need to perform a model comparison of each of the models shown.

2. Select the check boxes next to all the models shown at the top of the Results page. You can also select the check box next to **Champion** at the top of the table.

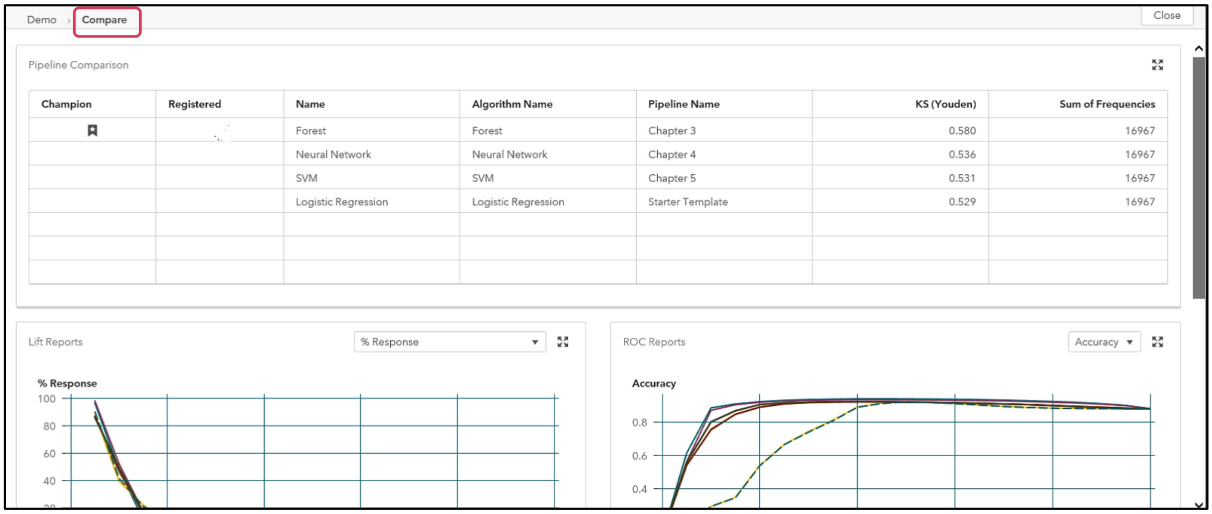


☑	Champion	Name
☑	🔖	Forest
☑		Neural Network
☑		SVM
☑		Logistic Regression

3. When multiple models are selected, the Compare button in the upper right corner is activated. Click the **Compare** button.

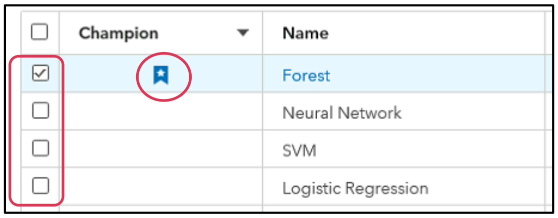


The Compare results are shown, where assessment statistics and graphics can be compared across all champion models from each of the pipelines.



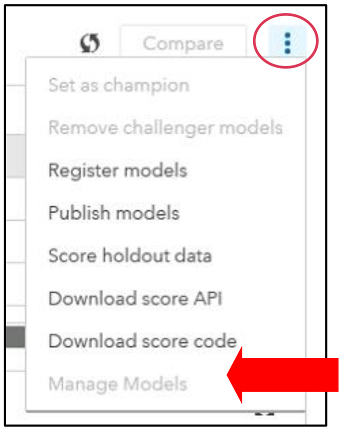
Champion	Registered	Name	Algorithm Name	Pipeline Name	KS (Youden)	Sum of Frequencies
★		Forest	Forest	Chapter 3	0.580	16967
		Neural Network	Neural Network	Chapter 4	0.536	16967
		SVM	SVM	Chapter 5	0.531	16967
		Logistic Regression	Logistic Regression	Starter Template	0.529	16967

4. Close the Compare results window and deselect models in the table at the top of the window until only the overall champion model is selected. The overall champion model is indicated with the star symbol in the **Champion** column.

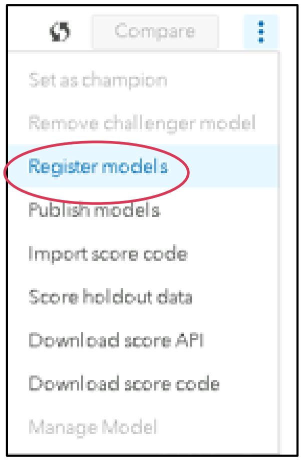


□	Champion	Name
☑	★	Forest
□		Neural Network
□		SVM
□		Logistic Regression

5. Click the three vertical dots on the right top corner to access the Project pipeline menu. Note that Manage Models is not available.



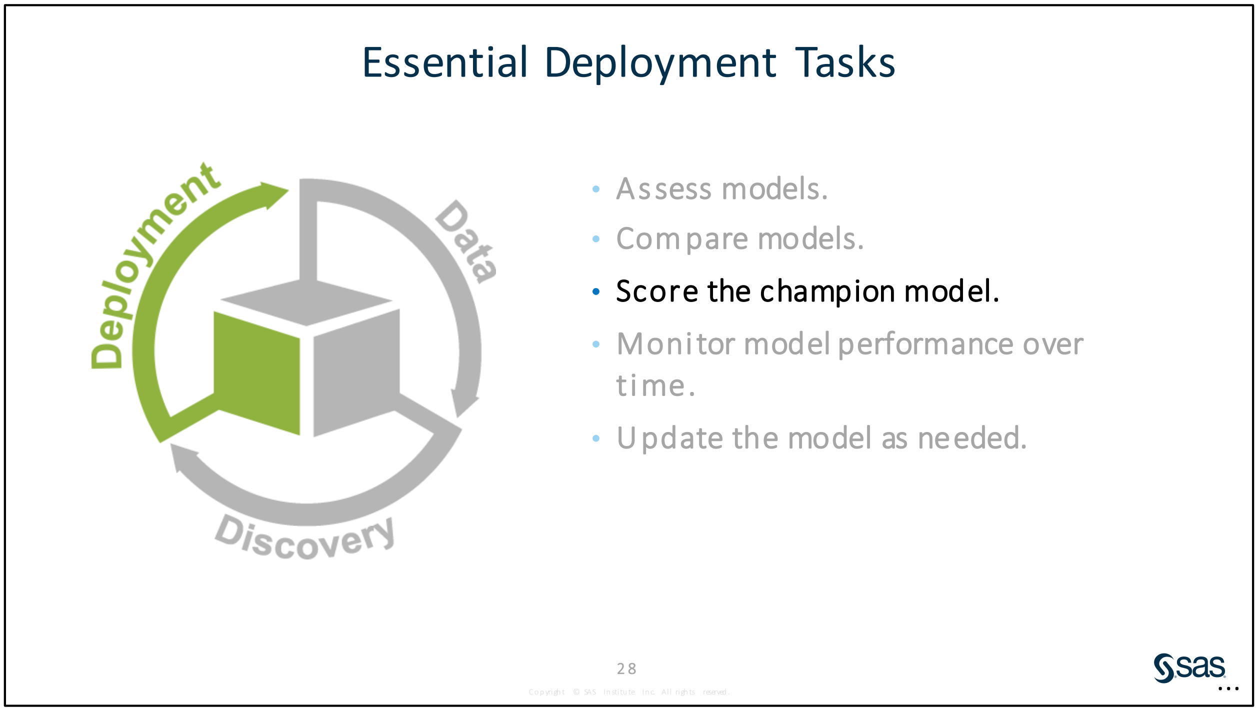
6. Select **Register models**. The champion model is registered.



When the champion model is registered, it can be viewed and used in the Manage Model tool, where you can export the score code in different formats, deploy the model, and manage its performance over time.

End of Demonstration

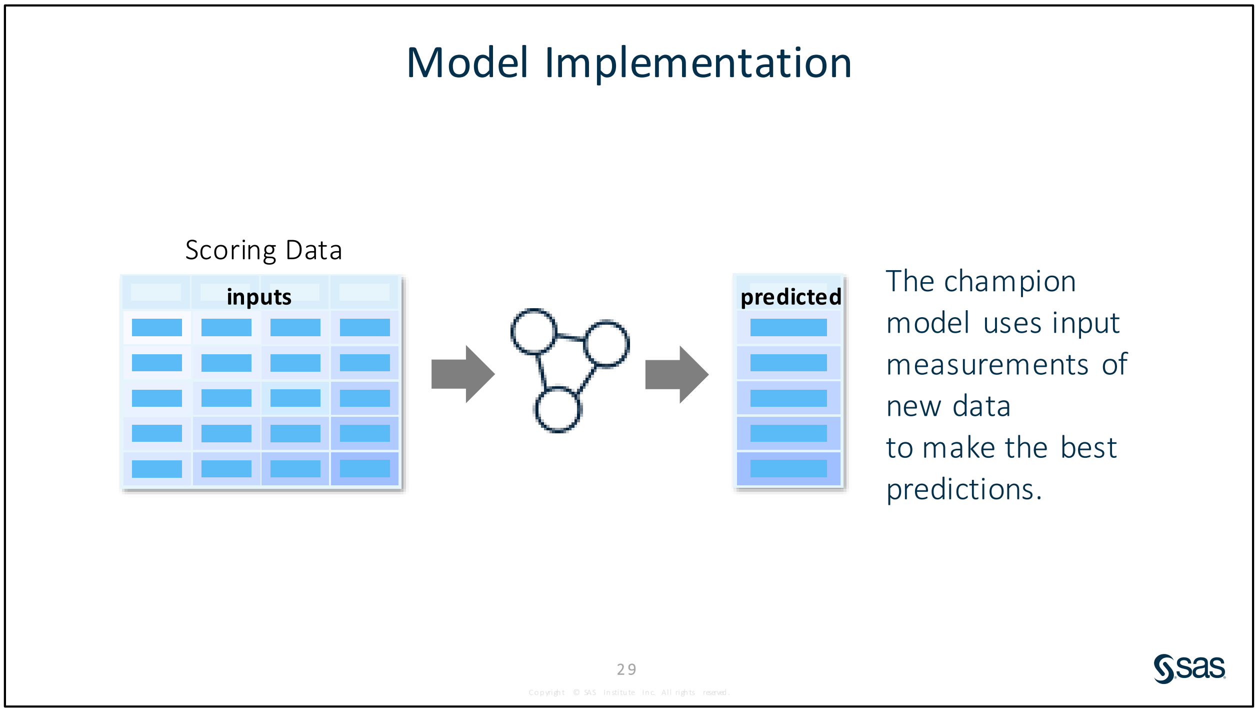
6.2 Model Deployment



Essential Deployment Tasks

- Assess models.
- Compare models.
- Score the champion model.
- Monitor model performance over time.
- Update the model as needed.

Scoring is the generation of predicted values for a data set that might not contain a target variable.



Model Implementation

Scoring Data

inputs

predicted

The champion model uses input measurements of new data to make the best predictions.

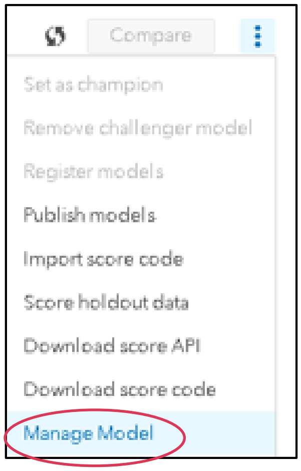
29

After you train and compare predictive models, one model is selected to represent the association between the inputs and the target. After it is selected, this model must be put to use. The model is translated to score code or a score model data set. It is then applied to the scoring data set, where predicted outcomes are obtained.

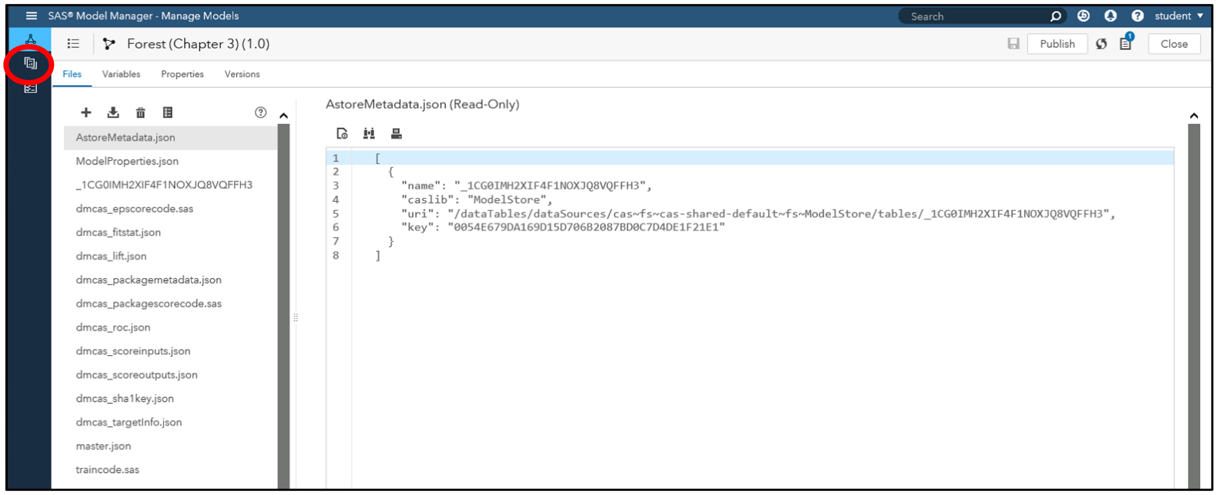
Model Studio Models in Model Manager

In this demonstration, you place the champion model selected by Pipeline Comparison in Model Studio into the Manage Models tool in SAS Viya.

1. In Pipeline Comparison, click the "snowman" icon (three vertical dots) to access the project pipeline menu. Notice that the Manage Model option is now available, after you have registered the champion model in the previous demonstration. Select **Manage Model**.



2. You are redirected to the Manage Model tool. A window containing a list of files is shown, which includes codes for training and scoring the model. Click the second icon assigned to the projects on the left pane of the window. This icon takes you to the Model Manager projects.

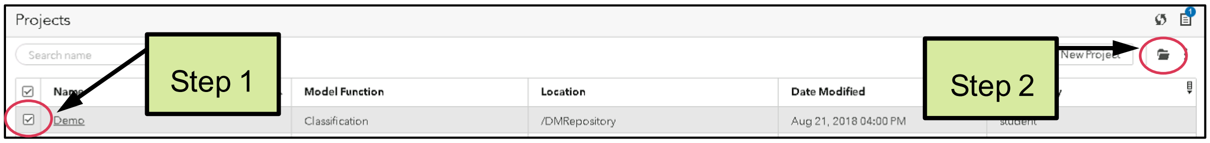


AstoreMetadata.json
ModelProperties.json
_1CG0IMH2XIF4F1NOXJQ8VQFFH3
dmcas_epscorecode.sas
dmcas_fitstat.json
dmcas_lift.json
dmcas_packagemetadata.json
dmcas_packagescorecode.sas
dmcas_roc.json
dmcas_scoreinputs.json
dmcas_scoreoutputs.json
dmcas_sha1key.json
dmcas_targetInfo.json
master.json
traincode.sas

AstoreMetadata.json (Read-Only)

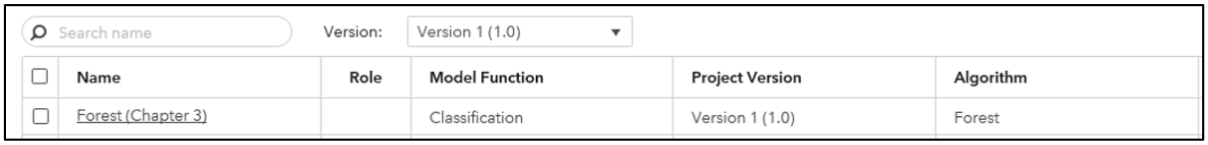
```
[
  {
    "name": "_1CG0IMH2XIF4F1NOXJQ8VQFFH3",
    "caslib": "ModelStore",
    "uri": "/dataTables/dataSources/cas~fs~cas-shared-default~fs~ModelStore/tables/_1CG0IMH2XIF4F1NOXJQ8VQFFH3",
    "key": "0054E679DA169D15D706B2087BD0C7D4DE1F21E1"
  }
]
```

3. The Model Manager project named Demo is based on the Model Studio project of the same name. Also in this project are models that were registered within Model Studio. Select the **Demo** project (step 1) and click the open shortcut button in the upper right corner (step 2), or click the project name.



☑	Name	Model Function	Location	Date Modified	
☑	Demo	Classification	/DMRepository	Aug 21, 2018 04:00 PM	student

4. Models within the Demo project that have been registered are shown.



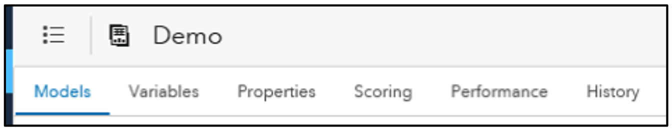
Search name

Version: Version 1 (1.0) ▾

☐	Name	Role	Model Function	Project Version	Algorithm
☐	Forest (Chapter 3)		Classification	Version 1 (1.0)	Forest

There is currently a single model in the Demo project, a forest. The name of the model, Forest (Chapter 3), is based on the champion model and pipeline name from Model Studio.

Across the top of the page is a series of tabs. These tabs are used during the entire model management process, which goes beyond just model deployment.



The Models tab shows registered models within the Model Manager project.

The Variables tab is where input variables and output variables can be added to both project and model objects.

The Properties tab contains the project metadata. Project metadata includes information such as the name of the project, the type of project, the project owner, the project location, and which tables and variables are used by project processes, such as scoring. Project properties are organized into General, Tags, and User Defined.

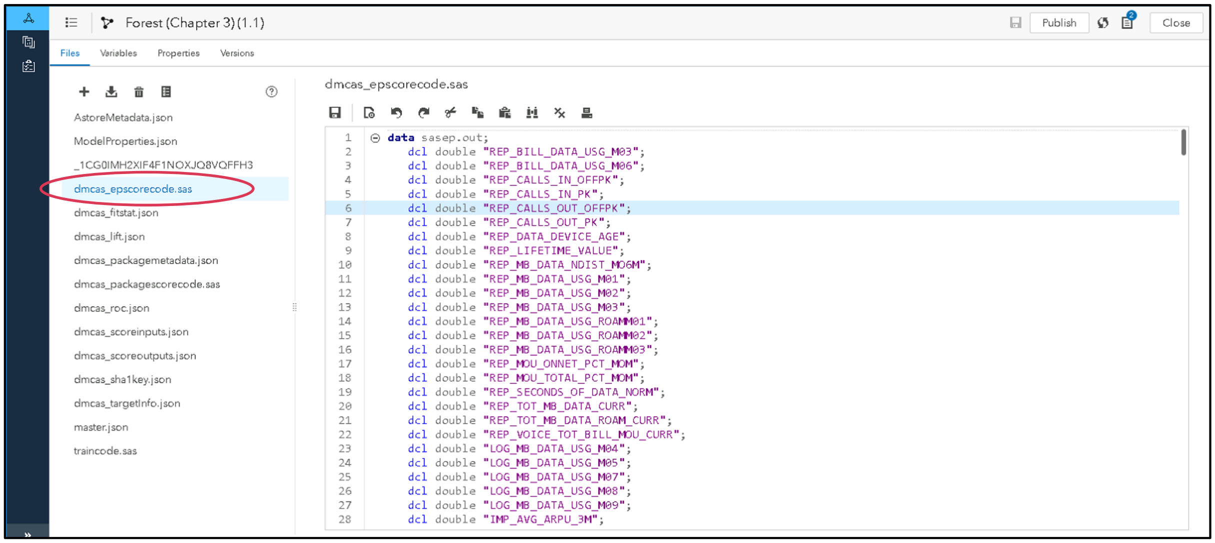
The Scoring tab is where scoring tests can be run and also where published models can be validated.

The Performance tab shows performance monitoring reports, which are generated from scored data.

The History tab shows the history of how the project and model have been used, including information about when the project and models were created, when champion models were defined, and when the model was last deployed.

5. Click the name of the model.

6. Select **dmcas_epscorecode.sas** in the column on the left.

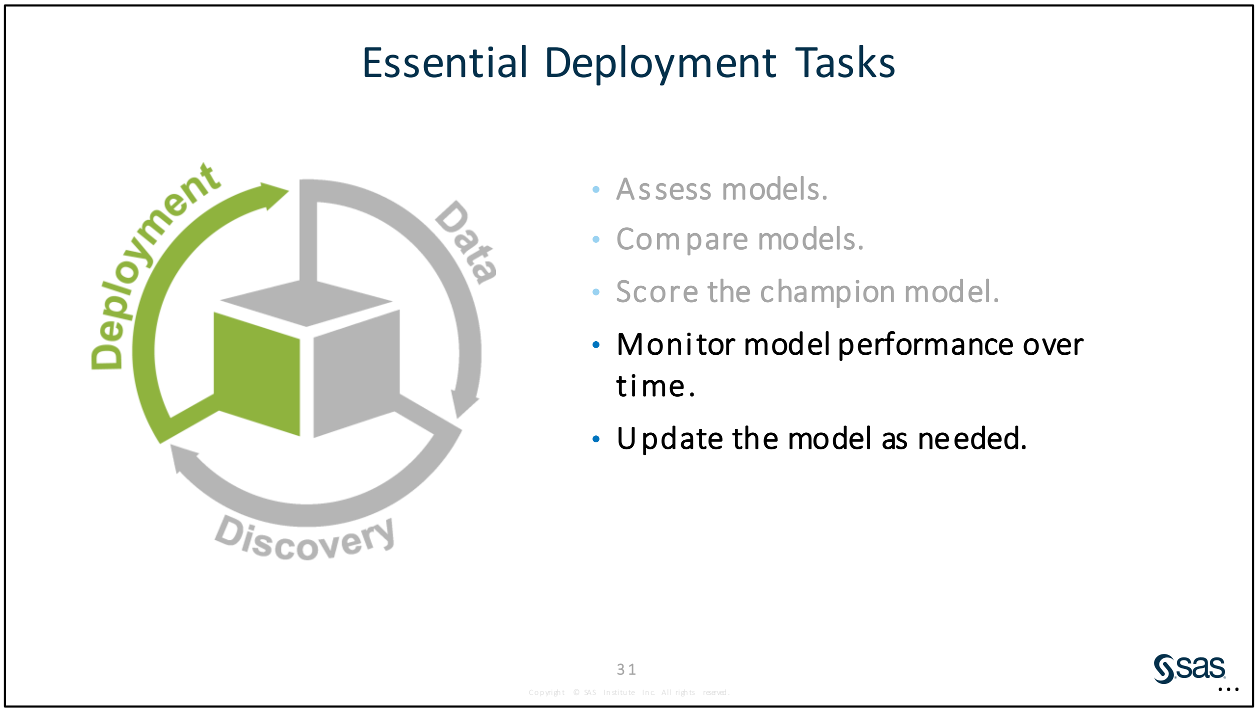


```
data sasep.out;
   dcl double "REP_BILL_DATA_USG_M03";
   dcl double "REP_BILL_DATA_USG_M06";
   dcl double "REP_CALLS_IN_OFFPK";
   dcl double "REP_CALLS_IN_PK";
   dcl double "REP_CALLS_OUT_OFFPK";
   dcl double "REP_CALLS_OUT_PK";
   dcl double "REP_DATA_DEVICE_AGE";
   dcl double "REP_LIFETIME_VALUE";
   dcl double "REP_MB_DATA_NDIST_MO6M";
   dcl double "REP_MB_DATA_USG_M01";
   dcl double "REP_MB_DATA_USG_M02";
   dcl double "REP_MB_DATA_USG_M03";
   dcl double "REP_MB_DATA_USG_ROAMM01";
   dcl double "REP_MB_DATA_USG_ROAMM02";
   dcl double "REP_MB_DATA_USG_ROAMM03";
   dcl double "REP_MOU_ONNET_PCT_MOM";
   dcl double "REP_MOU_TOTAL_PCT_MOM";
   dcl double "REP_SECONDS_OF_DATA_NORM";
   dcl double "REP_TOT_MB_DATA_CURR";
   dcl double "REP_TOT_MB_DATA_ROAM_CURR";
   dcl double "REP_VOICE_TOT_BILL_MOU_CURR";
   dcl double "LOG_MB_DATA_USG_M04";
   dcl double "LOG_MB_DATA_USG_M05";
   dcl double "LOG_MB_DATA_USG_M07";
   dcl double "LOG_MB_DATA_USG_M08";
   dcl double "LOG_MB_DATA_USG_M09";
   dcl double "IMP_AVG_ARPU_3M";
```

Above is the score code generated by the model. The score code can be exported to deploy the model in production considering distinct environments and platforms.

Note: For more information about model deployment using Model Manager or about Model Manager in general, consider taking further training on the product.

End of Demonstration



Essential Deployment Tasks

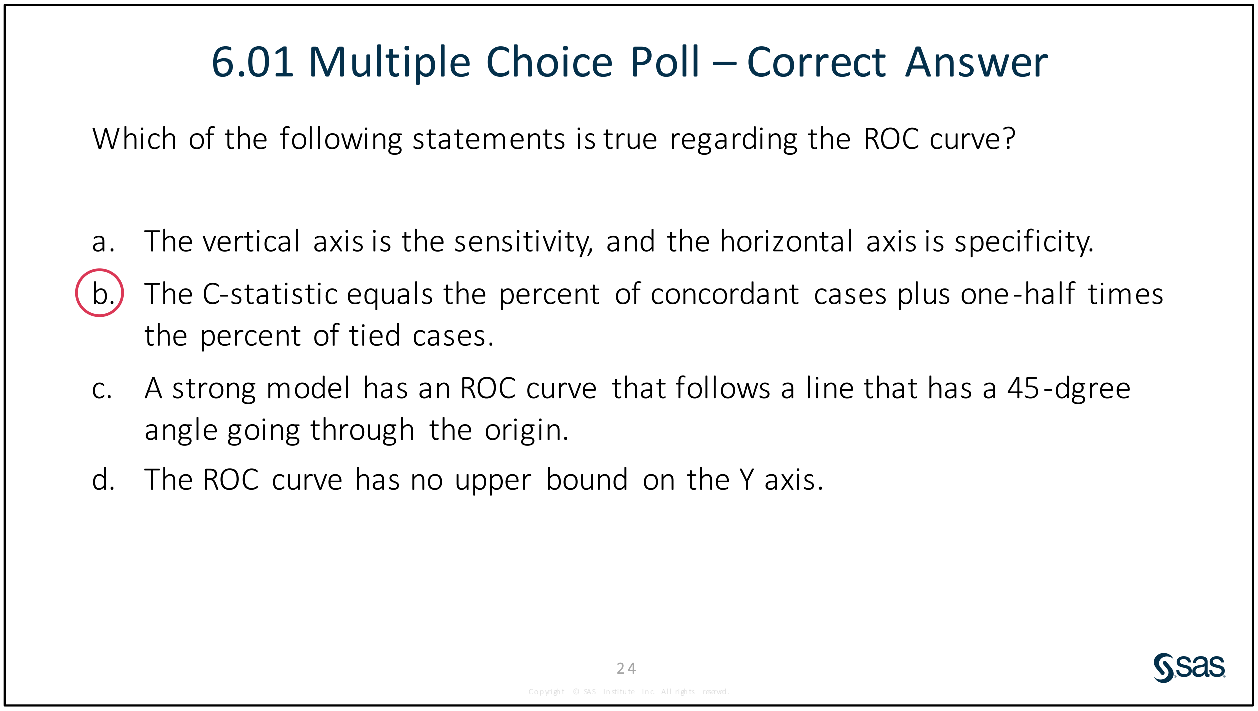
- Assess models.
- Compare models.
- Score the champion model.
- Monitor model performance over time.
- Update the model as needed.

31

Even after a model has been deployed, it must be monitored. Because models are often trained on static snapshots of data, their predictions typically become less accurate over time as the environment shifts away from the conditions that were captured in the training data. For example, consider a movie recommendation model that must adapt as viewers grow and mature through stages of life. After a certain period, the error rate on new data surpasses a predefined threshold, and models must be retrained or replaced. Champion-challenger testing is another common model deployment practice, in which a new, challenger model is compared against a currently deployed model at regular time intervals. When a challenger model outperforms a currently deployed model, the deployed model is replaced by the challenger, and the champion-challenger process is repeated. Yet another approach to refreshing a trained model is through online updates. Online updates continuously change the value of model parameters or rules based on the values of new, streaming data. It is prudent to assess the trustworthiness of real-time data streams before implementing an online modeling system.

6.3 Solutions

Solutions to Student Activities (Polls/Quizzes)



6.01 Multiple Choice Poll – Correct Answer

Which of the following statements is true regarding the ROC curve?

a. The vertical axis is the sensitivity, and the horizontal axis is specificity.

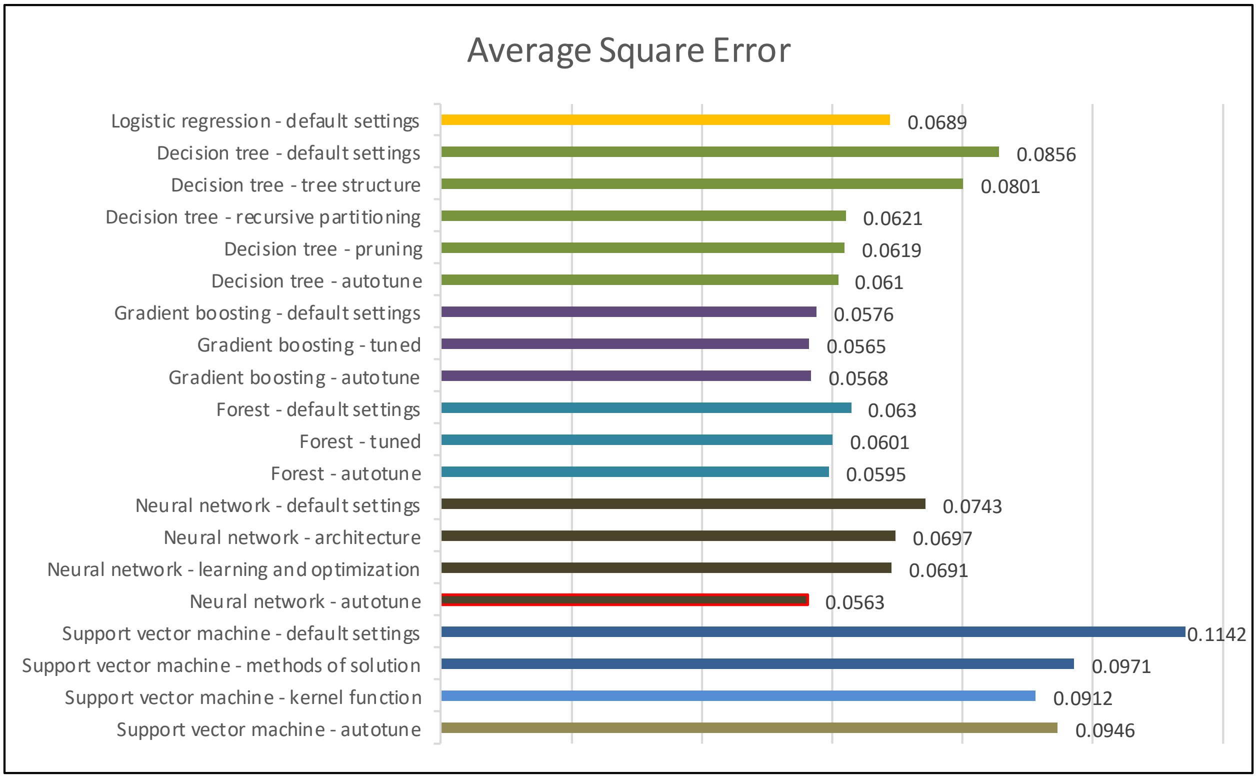
[✓ b.] The C-statistic equals the percent of concordant cases plus one-half times the percent of tied cases.

c. A strong model has an ROC curve that follows a line that has a 45-dgree angle going through the origin.

d. The ROC curve has no upper bound on the Y axis.

Summary of All Models

The following chart shows the different models' performance for logistic regression, decision tree, gradient boosting, forest, neural network, and support vector machine. The comparison of these models considers the default settings, the tune based on the topics covered throughout the chapters, and the Autotune feature for each technique.



The Neural Network model autotuned during the exercise was the best model.

Appendix A References

A.1 References

Alpaydin, E. and F. Gürgen. 1996. "Comparison of Statistical and Neural Classifiers and Their Applications to Optical Character Recognition and Speech Classification." in *Neural Network Systems: Techniques and Applications* (Leondes, ed.). San Diego, CA: Academic Press.

Banks, R. B. 1994. *Growth and Diffusion Phenomena: Mathematical Frameworks and Application*. New York: Springer-Verlag.

Bartlett, P. L. 1997. "For Valid Generalization, the Size of the Weights is More Important than the Size of the Network." in: *Advances in Neural Information Processing Systems Volume 9* (Mozer, Jordan, and Petsche, eds.). Cambridge, MA: The MIT Press.

Bauer, E. and R. Kohavi. 1999. "An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants." *Machine Learning*. 36:105–139.

Beck, A. 1997. "Herb Edelstein discusses the usefulness of data mining." DS Star. Vol. 1:No. 2. Available at www.tgc.com/dsstar/.

Berger, J. 1980. *Statistical Decision Theory*. New York: Springer-Verlag.

Berk, K. N. and D. E. Booth. 1995. "Seeing a Curve in Multiple Regression." *Technometrics* 37:4.

Bishop, C. M. 1995. Neural Networks for Pattern Recognition. New York: Oxford University Press.

Blake, C., E. Keogh, and C. J. Merz. 1998. UCI Repository of machine learning databases. Irvine, CA: University of California, Department of Information and Computer Science. Available at http://archive.ics.uci.edu/ml/

Box, G. E. P. and G. M. Jenkins. 1976. *Time Series Analysis: Forecasting and Control*. San Francisco: Holden-Day Inc.

Breiman, L. et al. 1984. *Classification and Regression Trees*. Belmont, CA: Wadsworth International Group.

Breiman, L. 1996a. "Technical Note: Some Properties of Splitting Criteria." *Machine Learning*. 24:41–47.

Breiman, L. 1996b. "Bagging Predictors." *Machine Learning*. 24:123–140.

Breiman, L. 1998. "Arcing Classifiers (with discussion)." *Annals of Statistics*. 26:801–849.

[BFOS], Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees*. New York: Chapman & Hall.

Broomhead, D. S. and David Lowe. 1988. *Radial basis functions, multi-variable functional interpolation and adaptive networks* (Technical report). RSRE. 4148.

Broyden, C. G. 1970 "The Convergence of a Class of Double-rank Minimization Algorithms." *Journal of the Institute of Mathematics and Its Applications* 6:76–90.

Byrd, R. H., P. Lu, J. Nocedal, and C. Zhu. 1995. "A limited memory algorithm for bound constrained optimization." *SIAM Journal on Scientific Computing*. 16, 1190-1208.

Bryson, A. E. and Y. C. Ho. 1969. *Neural Networks: Computers with Intuition*. Singapore: World Scientific.

Cai, Z. and C. L. Tsai. 1999. "Diagnostics for Nonlinearity in Generalized Linear Models." *Computational Statistics and Data Analysis* 29:445–469.

Carroll, R. J. and D. Ruppert. 1988. *Transformation and Weighting in Regression*. New York: Chapman & Hall.

Chatfield, C. and J. Faraway. 1998. "Time Series Forecasting with Neural Networks: A Comparative Study Using the Airline Data." *Applied Statistics* 47:231–250.

Chester, D.L.1990. "Why Two Hidden-Layers are Better than One" in IJCNN-90-WASH-DC. Lawrence Erlbaum. Vol. 1:265–268.

Cuellar, M.P., M. Delgado, and M.C. Pegalajar. 2006. "An Application of Non-Linear Programming to Train Recurrent Neural Networks in Time Series Prediction Problems." Enterprise Information Systems VII (Springer Netherlands) 95–102.

Cybenko, G. 1988. *Continuous Valued Neural Networks with Two Hidden Layers Are Sufficient*. Technical Report. Dept. of Computer Science. Tufts University: Medford, MA.

David Shepard Associates. 1999. *The New Direct Marketing: How to Implement a Profit-Driven Database Marketing Strategy*. New York: McGraw-Hill.

De Vries, B. and J. C. Principe. 1992. "The Gamma Filter - A New Class of Adaptive IIR Filters with Restricted Feedback." *IEEE Transactions*. Signal Processing. In Press.

De Vries, B. and J. C. Principe. 1992. "The Gamma Model - A New Model for Temporal Processing." *Neural Networks* Vol.5:565–576.

De Vries, B. and J. C. Principe. 1992. *Short Term Memory Structures for Dynamic Neural Networks*. Princeton, NJ: David Sarnoff Research Center.

Dorffner, G. 1996. "Neural Networks for Time Series Processing." *Neural Network World* 6:447–468.

Efron, B. 1983. "Estimating the Error Rate of a Prediction Rule: Improvement on Cross Validation." *Journal of the American Statistical Association* 78:316–331.

Efron, B., T. Hastie, I. Johnstone, and R. Tibshirani. 2004. "Least Angle Regression (with Discussion)." *Annals of Statistics*, 32: 407–499.

Ellingsen, B.K. 1994. "A Comparative Analysis of Backpropagation and Counterpropagation Neural Networks." Department of Information Science. University of Bergen: Bergen, Norway.

Elman, J.L. 1990. "Finding Structure in Time." *Cognitive Science* 14:179–221.

Ezekiel, M. 1924. "A Method for Handling Curvilinear Correlation for any Number of Variables." *Journal of the American Statistical Association* 19:431–453.

Falhman, S. E. 1988 "Faster-learning variations on back-propagation: an empirical study." *Proceedings of the 1988 Connectionist Models Summer School*. (D. Touretzky, G. E. Hinton, and T. J. Sejnowski, eds.).pp. 31–51. San Mateo, CA. Morgan Kaufmann.

Fahlman, S. E. and C. Lebiere. 1990. "The Cascade-Correlation Learning Architecture." in *Advances in Neural Information Processing Systems Volume 2*. (D Touretzky, ed.). San Mateo, CA: Morgan Kaufmann.

Fan, J. and I. Gijbels. 1996. *Local Polynomial Modeling and its Applications*. New York: Chapman & Hall.

Fletcher, R. 1970. "A New Approach to Variable Metric Algorithms." *Computer Journal* 13: 317–322.

Fletcher, R. 1987. *Practical Methods of Optimization*. New York: Wiley.

Freund, Y. and R. E. Schapire. 1996. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting." *Journal of Computer and System Science*. 55:119–139.

Friedman, J. H. 1991. "Multivariate Adaptive Regression Splines (with discussion)." *Annals of Statistics* 19:1–141.

Friedman, J. H. 1994. "An Overview of Predictive Learning and Function Approximation." In *From Statistics to Neural Networks. Theory and Pattern Recognition Applications*. (Cherkasy, Friedman, Wechsler, eds.) New York: Springer-Verlag.

Friedman, J. H. 1997. "On Bias, Variance, 0/1-Loss, and the Curse-of-Dimensionality." *Data Mining and Knowledge Discovery* 1:55–77.

Friedman, J. H. 2001. "Greedy function approximation: A gradient boosting machine." *The Annals of Statistics*. 29:1189–1232.

Friedman, J. H. 2002. "Stochastic gradient boosting." *Computational Statistics & Data Analysis*. 38:367–378.

Friedman, J. H. and W. Stuetzle 1981. "Projection Pursuit Regression." *Journal of the American Statistical Association*. 76:817–823.

Furnival, G. M. and R. W. Wilson. 1974. "Regression by Leaps and Bounds." *Technometrics*, 16, 499–511.

Georges, J. E. 2003. "Beyond Expectations: Quantifying Variability in Predictive Models." *Proceedings of the M2003 SAS Data Mining Conference*. Cary, NC: SAS Institute Inc.

Georges, J. E. 2004. "Qualities to Quantities: Using Non-numeric Data in Parametric Prediction." *Proceedings of the M2004 SAS Data Mining Conference*. Cary, NC: SAS Institute Inc.

Goldfarb, D. 1970. "A Family of Variable Metric Updates Derived by Variational Means." *Mathematics of Computation* 24: 23–24.

Hand, D. J. 1997. *Construction and Assessment of Classification Rules*. New York: Wiley.

Hand, D. J. 2005. "What you get is what you want? – Some dangers of black box data mining." *M2005 Conference Proceedings*. Cary, NC: SAS Institute Inc.

Hand, D. J. 2006. "Classifier technology and the illusion of progress." *Statistical Science* 21:1–14.

Hand, D. J. and W. E. Henley. 1997. "Statistical classification methods in consumer credit scoring: a review." *Journal of the Royal Statistical Society A*. 160:523–541.

Hand, David, Heikki Mannila, and Padraic Smyth. 2001. *Principles of Data Mining*. Cambridge, Massachusetts: The MIT Press.

Harrell F. E., K. L. Lee, and D. B. Mark. 1996. "Multivariate Prognostic Models: Issues in Developing Models, Evaluating Assumptions and Adequacy, and Measuring and Reducing Errors." *Statistics in Medicine* 15:361–387.

Harrell, F. E. 2006. *Regression Modeling Strategies*. New York: Springer-Verlag New York, Inc.

Harrison, D. and D. L. Rubinfeld. 1978. "Hedonic Prices in the Demand for Clean Air." *Journal of Environmental Economics and Management* 5:81–102.

Hassibi, B. and D. G. Stork. 1993. "Second Order Derivatives for Network Pruning: Optimal Brain Surgeon." In *Advances in Neural Information Processing Systems, Volume 5*. (Hansen, Cowan, and Giles, eds.) San Mateo, CA: Morgan Kaufmann.

Hastie, T. J. and R. J. Tibshirani. 1986. "Generalized Additive Models (with discussion)." *Statistical Science* 1:297–318.

Hastie, T. J. and R. J. Tibshirani. 1990. *Generalized Additive Models*. New York: Chapman & Hall.

Hastie, T. J. and R. J. Tibshirani, and Jerome Friedman. 2001. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York: Springer-Verlag New York, Inc.

Hebb, D. O. 9949. *The Organization of Behavior*. New York: Wiley.

Hecht-Nielsen, R. 1987. "Counterpropagation Networks." *Applied Optics* 26:4979–4984.

Hertz, J., A. Krogh, and R. G. Palmer. 1991. *Introduction to the Theory of Neural Computation*. Redwood City, CA: Addison-Wesley Publishing Co.

Hettich, S. and S. D. Bay. 1999. The UCI KDD Archive [<http://kdd.ics.uci.edu>]. Irvine, CA: University of California, Department of Information and Computer Science.

Hinton, G. E., S. Osindero, and Y. W. Teh. 2006. "A Fast Learning Algorithm for Deep Belief Networks." *Neural Computation 18*. MIT. 1527–1554.

Hinton, G. E., N. Shrivastava, and K. Swersky. 2013. "Overview of Mini-Batch Gradient Descent" Video from Coursera - University of Toronto - Course: Neural Networks for Machine Learning: Published on Nov 5, 2013. Available at https://www.coursera.org/course/neura.

Hoaglin, D. C., F. Mosteller, and J. W. Tukey. 1983. *Understanding Robust and Exploratory Data Analysis*. New York: Wiley.

Hoerl, A. E. and R. W. Kennard. 1970. "Ridge Regression: Biased Estimation for Nonorthogonal Problems." *Technometrics* 12:55–67.

Hogg, R. V. 1979. "Statistical Robustness: One View of Its Use in Applications Today." *The American Statistician* 33:108–115.

Holden, K. 1995. "Vector Autoregression Modeling and Forecasting." *Journal of Forecasting* 14:159–166.

Holland, J. H. 1976. "Studies of Spontaneous Emergence of Self-Replicating Systems Using Cellular Automata and Formal Grammars." In *Automata, Languages, Development*. (A. Lindenmayer and G. Rozenberg, eds.) Amsterdam: North-Holland.

Holland, J. H. 1992. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, 2nd *Edition*. Cambridge, MA: The MIT Press.

Huber, P. J. 1964. "Robust Estimation of a Location Parameter." *Annals of Mathematical Statistics* 35:73–101.

Jacobs, R. A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton. 1991. "Adaptive Mixture of Local Experts." *Neural Computation* 3:79–87.

Jaeger, H. 2001. "The 'echo state' approach to analyzing and training recurrent neural networks". *GMD Report 148*. GMD - German National Research Institute for Computer Science.

Jaeger, H. 2002. "Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach". *GMD Report 159*. Fraunhofer Institute AIS.

Jaeger, H. 2007. "Echo state network." *Scholarpedia* 2(9):2330.

Johnson, D. E. 1998. *Applied Multivariate Methods for Data Analysis*. Pacific Grove, CA: Druxbury Press.

Johnson, G. 1991. *In the Palaces of Memory*. Vintage Books: Random House, Inc. NY, NY.

Jordan, M. I. and R. A. Jacobs. 1994. "Hierarchical Mixture of Experts and the EM algorithm." *Neural Computation* 6:181–214.

Kass, G. V. 1980. "An exploratory technique for investigating large quantities of categorical data." *Applied Statistics*. 29:119–127.

Kohavi, R., D. Sommerfield, and J. Dougherty. 1996. "Data Mining using MLC." *International Journal on Artificial Intelligence Tools*. 6:234–245.

Landwehr, J. D., D. Pregibon, and A. C. Shoemaker. 1984. “Graphical Methods for Assessing Logistic Regression Models (with discussion).” *Journal of the American Statistical Association* 79:61–83.

Larsen, W. A. and S. J. McCleary. 1972. “The Use of Partial Residual Plots in Regression Analysis.” *Technometrics* 14:781–790.

LeCun, Y., P. Y. Simard, and B. Pearlmutter. 1993. “Autonomic learning rate maximization by on-line estimation of Hessian’s eigenvectors” In *Advances in Neural Information Processing Systems* (S. J. Hanson, J. D. Cowan, and C. L. Giles, eds.) Vol. 5:156–163. San Mateo, CA: Morgan Kaufmann.

LeCun Y., L. Bottou, G. B. Orr, and K. R. Müller. 1998. “Efficient BackProp.” In *Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science* (G. B. Orr and K. R. Müller, eds). Vol. 1524. Berlin: Springer,

Leisch, F., A. Trapletti, and K. Hornik. 1999. “Stationarity and Stability of Autoregressive Neural Network Processes.” In *Advances in Neural Information Processing Systems 11. (*Kearns, Solla, Cohn, eds.) Cambridge, MA: The MIT Press.

Loh, W. and Y. Shih. 1997. “Split Selection Methods for Classification Trees.” *Statistica Sinica*. 7:815–840.

Loh, W. and N. Vanichsetakul. 1988. “Tree-Structured Classification Via Generalized Discriminant Analysis (with discussion).” *Journal of the American Statistical Association*. 83:715–728.

Lukosevicius, M. and H. Jaeger. 2010. “Reservoir Approaches to Recurrent Neural Network Training.” *Computer Science Revue* 3(3):127–149.

Lukosevicius, M. 2012. “A Practical Guide to Applying Echo State Networks.” In *Neural Networks: Tricks of the Trade, 2nd ed*. (G. Montavon, G. B. Orr, and K. R. Müller, eds.) Springer LNCS 7700:659–686.

Maass W., T. Natschlaeger, and H. Markram. 2002. “Real-time computing without stable states: A new framework for neural computation based on perturbations.” *Neural Computation*. 14(11):2531–2560.

Maldonado, M., J. Dean, W. Czika, and S. Haller. 2014. “Leveraging Ensemble Models in SAS® Enterprise Miner™.” *Proceedings of the SAS Global Forum 2014 Conference*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings14/SAS133-2014.pdf.

Masters, T. 1993. *Practical Neural Network Recipes in C++*. Boston: Academic Press Inc. Harcourt Brace & Co. Publishers.

Melssen, W., R. Wehrens, and L. Buydens. 2006. “Supervised Kohonen networks for classification problems.” *Chemometrics and Intelligent Laboratory Systems* 83:99–113.

McCullagh, P. and J. A. Nelder. 1989. *Generalized Linear Models, Second Edition*. New York: Chapman & Hall.

McLachlan, G. J. 1992. *Discriminant Analysis and Statistical Pattern Recognition*. New York: Wiley.

Minsky, Marvin. 1986. *The Society of Mind.* New York: Simon and Schuster.

Mitra, S. 2000. “Neuro-Fuzzy Rule Generation: Survey in Soft Computing Framework.” *IEEE Transactions on Neural Networks*. Vol. II, No 3 (May).

Moody, J. 1994. “Prediction Risk and Architecture Selection for Neural Networks.” In *From Statistics to Neural Networks. Theory and Pattern Recognition Applications*. (Cherkasy, Friedman, Wechsler, eds.) New York: Springer-Verlag.

Morgan, J. N. and J. A. Sonquist. 1963. “Problems in the Analysis of Survey Data, and a Proposal.” *Journal of the American Statistical Association*. 58:415–434.

Mosteller, F. and J. W. Tukey. 1977. *Data Analysis and Regression*. Reading, MA: Addison-Wesley.

Murthy, S. K., S. Kasif, and S. Salzberg. 1994. “A System of Induction of Oblique Decision Trees.” *Journal of Artificial Intelligence Research*. 2:1–32.

Murthy, S. K. and S. Salzberg. 1995. “Lookahead and Pathology in Decision Tree Induction.” *Proceedings of IJCAI-95*, *Montreal*. 1025–1031.

NC Health and Human Services. North Carolina State Center for Health Statistics. “Statistics and Reports.” Available at http://www.schs.state.nc.us/data/archivedvitalstats.cfm#vol1.

Ng, A. 2013. “Stochastic Gradient Descent” Video from Coursera - Stanford University - Course: Machine Learning: Published on Nov 1, 2013. Available at https://www.coursera.org/course/ml.

Olah, C. “Understanding LSTM Networks.” Available at http://colah.github.io/posts/2015-08-Understanding-LSTMs. Posted Aug 27, 2015. Accessed March 7, 2016.

Pao, Y. 1989. *Adaptive Pattern Recognition and Neural Networks*. Reading, MA: Addison-Wesley.

Parker, D. B. 1985. *Learning Logic: Technical Report TR-47*. Center for Computational Research in Economics and Management Science. MIT. Cambridge, MA.

Patil, G. P. and C. Taillie. 1982. “Diversity as a Concept and its Measurement (with discussion).” *Journal of the American Statistical Association*. 77:548–567.

Pearlmutter, B. 1995. “Gradient calculations for dynamic recurrent neural networks: A survey.” *Neural Networks.* IEEE Transactions. 6(5):1212–1218.

Piatesky-Shapiro, G. 1998. “What Wal-Mart might do with Barbie association rules.” Knowledge Discovery Nuggets. 98:1. Available at http://www.kdnuggets.com/.

Poh, H., J. Yao, and T. Jasic. 1998. “Neural networks for the Analysis and Forecasting of Advertising and Promotion Impact.” *International Journal of Intelligent Systems in Accounting, Finance, and Management* 7:253–268.

Potts, W. J. E. 1999. “Generalized Additive Neural Networks.” In *KDD-99 Proceedings.* (Chaudhuri, Madigan, eds.). ACM.

Prechelt, L. 1996. “A Quantitative Study of Experimental Evaluations of Neural Network Learning Algorithms: Current Research Practice.” *Neural Networks* 9:457–462.

Prechelt, L. 1997. “Investigation of the CasCor Family of Learning Algorithms.” *Neural Networks* 10:885–896.

Prechelt, L. 1998. “Automatic Early Stopping Using Cross Validation: Quantifying the Criteria.” *Neural Networks* 11:761–767.

Principe, J.C., N. R. Euliano, and W. C. Lefebvre. 2000. *Neural and Adaptive Systems*. New York: Wiley.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Raftery, A. E. 1995. “Bayesian Model Selection in Social Research (with discussion).” In *Sociological Methodology 1995* (Marsden ed.) New York: Blackwell.

Rao, J. S. and W. J. E. Potts. 1997. “Visualizing Bagged Decision Trees.” *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining* (Heckerman, Mannila, Pregibon, and Uthurusamy, eds.). Menlo Park, CA: AAAI Press.

Rendle, S. 2010. “Factorization Machines. “ The Institute for Scientific and Industrial Research. Osaka University, Japan.

Rendle, S. 2012. "Factorization Machines with libFM." *ACM Transactions on Intelligent Systems and Technology*, vol. 3, no. 3, article 57.

Ridgeway, Greg. 1999. "The State of Boosting." *Computing Science and Statistics*. 31:171–181.

Riedmiller, M. and H. Braun. 1993. "A direct adaptive method for faster back propagation learning: The RPROP algorithm." *Proceedings of the International Conference on Neural Networks*. San Francisco, CA. pp. 586–591.

Riedmiller, M. 1994. "Supervised Learning in Multi-layer Perceptrons – From Backpropagation to Adaptive Learning Algorithms." *Int. Journal of Computer Standards and Interfaces* 16.

Ripley, B. D. 1996. *Pattern Recognition and Neural Networks*. New York: Cambridge University Press.

Rubinkam, M. 2006. "Internet Merchants Fighting Costs of Credit Card Fraud." AP Worldstream. The Associated Press.

Rud, Olivia Parr. 2001. *Data Mining Cookbook: Modeling Data, Risk, and Customer Relationship Management*. New York: Wiley.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams. 1986. "Learning Representations by Back-Propagating Errors." *Nature* 323:533–536.

Runkle, D. E. 1987. "Vector Autoregressions and Reality (with discussion)." *Journal of Business and Economic Statistics* 5:437–454.

Sarle, W. S. 1994a. "Neural Networks and Statistical Models." *Proceedings of the Nineteenth Annual SAS® Users Group International Conference*. Cary, NC: SAS Institute Inc. 1538–1550.

Sarle, W. S. 1994b. "Neural Network Implementation in SAS® Software." *Proceedings of the Nineteenth Annual SAS® Users Group International Conference*. Cary, NC: SAS Institute Inc. 1550–1573.

Sarle, W. S. 1995. "Stopped Training and Other Remedies for Overfitting." *Proceedings of the 27th Symposium on the Interface*.

Sarle, W. S., ed. 1997. Neural Network FAQ. URL: ftp://ftp.sas.com/pub/neural/FAQ.html.

Sarle, W. S. 1999. "How to Measure the Importance of Inputs?" URL: ftp://ftp.sas.com/pub/neural/importance.html.

Sarle, W. S. 2000. (revised June 23) "How to measure importance of inputs?" Cary, N.C. [http://mu.dmt.ibaraki.ac.jp/yanai/neu/faq/importance.html]

SAS Institute Inc. 2016. *SAS® Enterprise Miner 14.2: High-Performance Procedures*. Cary, NC: SAS Institute Inc.

Scheffe, H. 1959. *The Analysis of Variance*. New York: Wiley.

SCHS: North Carolina State Center for Health Statistics. 2012. "Selected Statistics for 2000 and 1996-00." Available at http://www.schs.state.nc.us/data/vital/volume1/2000/nc.html.

SCHS: North Carolina State Center for Health Statistics. 2003. "Selected Statistics for 2001 and 1997-2001." Available at http://www.schs.state.nc.us/data/vital/volume1/2001/nc.html.

Schwarz, G. 1978. "Estimating the Dimension of a Model." *Annals of Statistics* 6:461–464.

Seber, G. A. F. and C. J. Wild. 1989. *Nonlinear Regression*. New York: Wiley.

Shanno, D. F. 1970. "Conditioning of Quasi-Newton Methods for Function Estimation." *Mathematics of Computation* 24:647–656.

Shannon, C. E. 1948. “A Mathematical Theory of Communication.” *The Bell System Technical Journal*. 27:379–423.

Suykens, J., B. D. Moor, and J. Vandewalle. 2008. “Toward optical signal processing using photonic reservoir computing.” *Optics Express* 16(15):11182–11192.

Tibshirani, R. and G. E. Hinton. 1995 “Coaching variables for regression and classification.” *Statistics and Computing* 8:25–33.

Tsukimoto, H. 2000. “Extracting Rules from Trained Neural Networks.” *IEEE Transactions on Neural Networks*. Vol. II, No. 2 (March).

Vapnik, V. N. 1995. *The Nature of Statistical Learning*. New York, NY: Springer.

Vapnik, V., S. Golowich, and A. Smola. 1997. “Support Vector Method for Function Approximation, Regression Estimation, and Signal Processing.” In *Advances in Neural Information Processing System 9*. 281-287. Also in *Proceedings of the 1996 conference on Neural Information Processing Systems*. MIT Press.

Vracko, M. 2005. “Kohonen Artificial Neural Network and Counterpropagation Neural Network in Molecular Toxicity Studies.” *Current Computer-Aided Drug Design* 1:73–78.

Wasserman, P. D. 1993. *Advanced Methods in Neural Computing*. New York: Van Nostrand Reinhold.

Weigend, A. S. and N. A. Gershenfeld. 1994. *Time Series Prediction: Forecasting the Future and Understanding the Past*. Reading, MA: Addison-Wesley.

Weiss, S. M. and C. A. Kulikowski. 1991. Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems. San Mateo, CA: Morgan Kaufmann.

Werbos, P. 1974. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Ph.D. Thesis. Harvard University.

Wold, H. 1966. “Estimation of Principal Components and Related Models by Iterative Least Squares”. In Krishnaiaah, P.R. *Multivariate Analysis*. New York: Academic Press. 391–420.

Zahavi, J. and N. Levin. 1997. “Applying Neural Computing to Target Marketing.” *Journal of Direct Marketing* 11:5–22.

Zhou, Z.-H. 2012. *Ensemble Methods: Foundations and Algorithms*. Boca Raton, FL: Chapman & Hall/CRC.