1 How To Use This Document

Highly regulated industries, such as banking and insurance, must comply with government regulations for model validation before a model can be put into production. This includes creating robust model development documentation. DataRobot automates the generation of model documentation, expediting the process required for regulatory compliance and following best practice for reducing model risk.

This document is split into two components: those sections that are automatically produced by DataRobot and those that require further input by the user. The sections in blue italicized font include specific instructions for the documenter and require additional user input of organization-specific information, such as business use cases, data sources, and implementation details. Once the sections are complete, remove the instructions. The remaining sections in non-blue italicized font are automatically populated by DataRobot and require no further input.

Copyright ©2024, DataRobot, Inc.



Table of Contents

* 1 How To Use This Document
* 2 DataRobot Model Development Documentation
* 3 Executive Summary and Model Overview
* 3.1 Model Stakeholders
* 3.2 Model Development Purpose and Intended Use
* 3.3 Model Description and Overview
* 3.4 Overview of Model Results
* 3.5 Model Interdependencies
* 4 Model Data Overview
* 4.1 Feature Association
* 4.2 Data Source Overview and Appropriateness
* 4.3 Input Data Extraction, Preparation, and Quality & Completeness
* 4.4 Data Assumptions
* 5 Model Theoretical Framework and Methodology
* 5.1 Model Development Overview
* 5.2 Model Assumptions
* 5.3 Model Methodology
* 5.3.1 Adds new images to the dataset by transforming copies of existing images.
* 5.3.2 Image Pretrained CNN Featurizer
* 5.3.3 Image Features Post Processor
* 5.3.4 Keras Neural Network Multi Classifier
* 5.4 Literature Review and References
* 5.5 Alternative Model Frameworks and Theories Considered
* 5.6 Variable Selection
* 5.6.1 DataRobot Quantitative Analysis
* 5.6.2 Expert Judgement and Variable Selection
* 5.6.3 Final Model Variables
* 5.6.3.1 Model Features and Summary Statistics
* 6 Model Performance and Stability
* 6.1 Model Validation Stability
* 6.1.1 Cross Validation Scores
* 6.1.2 Data Partitioning Methodology
* 6.2 Model Performance (Sample Scores)
* 6.3 Sensitivity Testing and Analysis
* 6.3.1 Lift Chart
* 6.3.2 Key Relationships
* 6.3.3 Accuracy
* 7 Model Implementation and Output Reporting
* 7.1 Version Control

2 DataRobot Model Development Documentation

A key component of effective model risk management is sufficiently detailed documentation for model development, implementation, and use, so that reasonable parties unfamiliar with a model can understand how the model operates, its limitations, and its key assumptions. Additionally, model documentation should contain enough detail for an independent party (e.g., independent model validation) to replicate all aspects of the underlying modeling process.

The purpose of this document is not to be prescriptive in format and content, but rather to serve as a guide in creating sufficiently rigorous model development, implementation, and use documentation. The documentation should provide enough evidence to show that the components of the model work as intended, the model is appropriate for its intended business purpose, and that it is conceptually sound.

3 Executive Summary and Model Overview

3.1 Model Stakeholders

Describe the model's purpose and its intended business use. Describe all stakeholders of this model, including their role, line-of-business, and team. This should include stakeholders of model ownership, model development, model implementation, and model risk management.

Model Owner(s): The individual who owns the business risk addressed by the model and provides approval for the model to be used within the line-of-business or enterprise function.

Model Developer(s): The individual responsible for building new models with DataRobot or maintaining existing models.

Model User(s): Those teams who will use the model output as part of their ongoing business operations.

Model Validator(s): The validators are responsible for independent model review and approval prior to its first use.

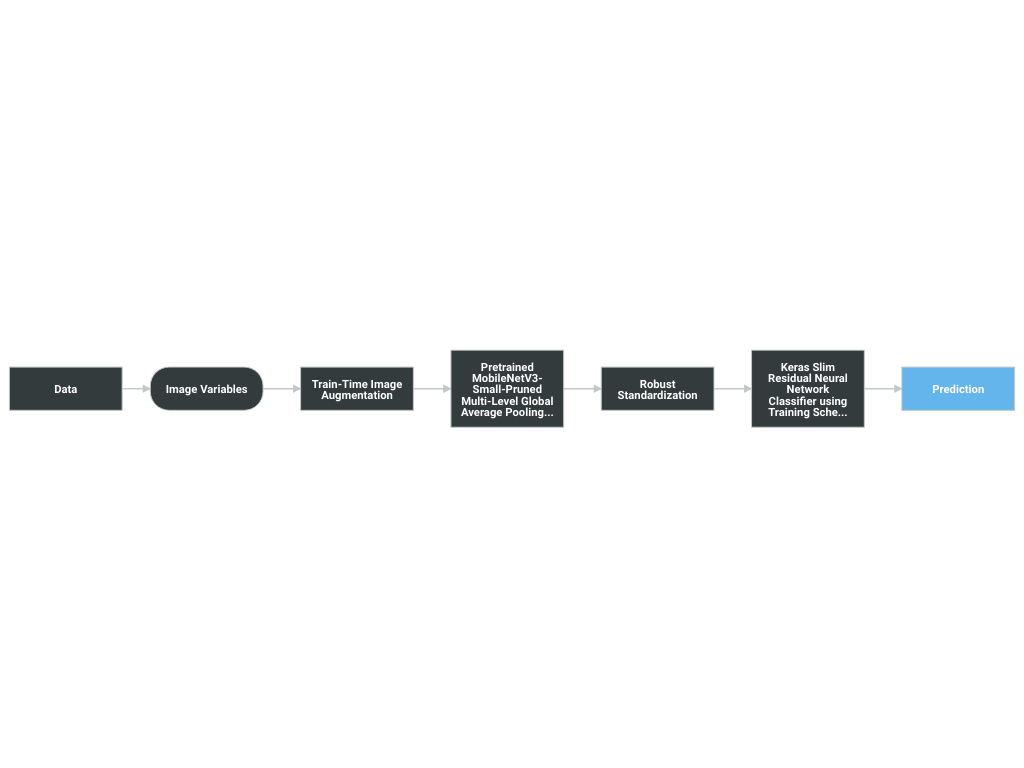
3.2 Model Development Purpose and Intended Use

Describe the model's purpose, including a summary of the business need for this particular model. Concisely describe how the model will be used to address this business problem. Furthermore, describe with great precision all model uses covered by this document. These descriptions will address this statement made in regulatory guidance, FRB SR-11-7, "Even a fundamentally sound model producing accurate outputs consistent with the design objective of the model may exhibit high model risk if it is misapplied or misused."

3.3 Model Description and Overview

The particular model referenced in this document: Keras Slim Residual Neural Network Classifier using Training Schedule (1 Layer: 64 Units). This model was developed in a project created with v7bfbfa5a7eba80ba of DataRobot. This model is denoted within DataRobot by the Project ID: 66b5df500edb94ae93e6b679 and the Model ID: 66b63b32fc283199d58ff7e2. The project was created on 2024-08-09 09:20:16.

The model development workflow process (i.e., the model blueprint) is detailed in the figure below.



A Blueprint represents the high-level end-to-end procedure for fitting the model, including any preprocessing steps, algorithms, and post-processing. It illustrates the many steps involved in transforming input predictors and targets into a model. Each element (or, “node”) in a blueprint can represent multiple steps.

The following elements connect to create the blueprint:

* Train-Time Image Augmentation
* Pretrained MobileNetV3-Small-Pruned Multi-Level Global Average Pooling Image Featurizer
* Robust Standardization
* Keras Slim Residual Neural Network Classifier using Training Schedule (1 Layer: 64 Units)

3.4 Overview of Model Results

DataRobot runs performance testing during the model development process to evaluate model results and reliability. The validation, cross-validation, and holdout (if applicable) out-of-sample performance scores are presented below, as well as the number of observations for each partition. The performance metric used for this project was LogLoss and the project included a total of 1,570 observations. An asterisk (\*) next to a score, whether validation or holdout, indicates that DataRobot used in-sample predictions to derive the score. (In-samples predictions are those that include data from the validation or holdout partitions due to sample size used to build the model.)

|  |  |
| --- | --- |
| Scoring Type | Score (LogLoss) |
| cross\_validation | 0.4507 |
| holdout | 0.5989 |
| validation | 0.3961 |

3.5 Model Interdependencies

Understanding interdependent relationships allows for enhanced understanding and improved ability to manage and aggregate model risk. Explain how this model is interconnected with other models in the model inventory--that is, the relative direction with regard to the model's position in the receiving input/sending output role. In addition to the directional relationship, also provide a brief description of each interconnected model.

4 Model Data Overview

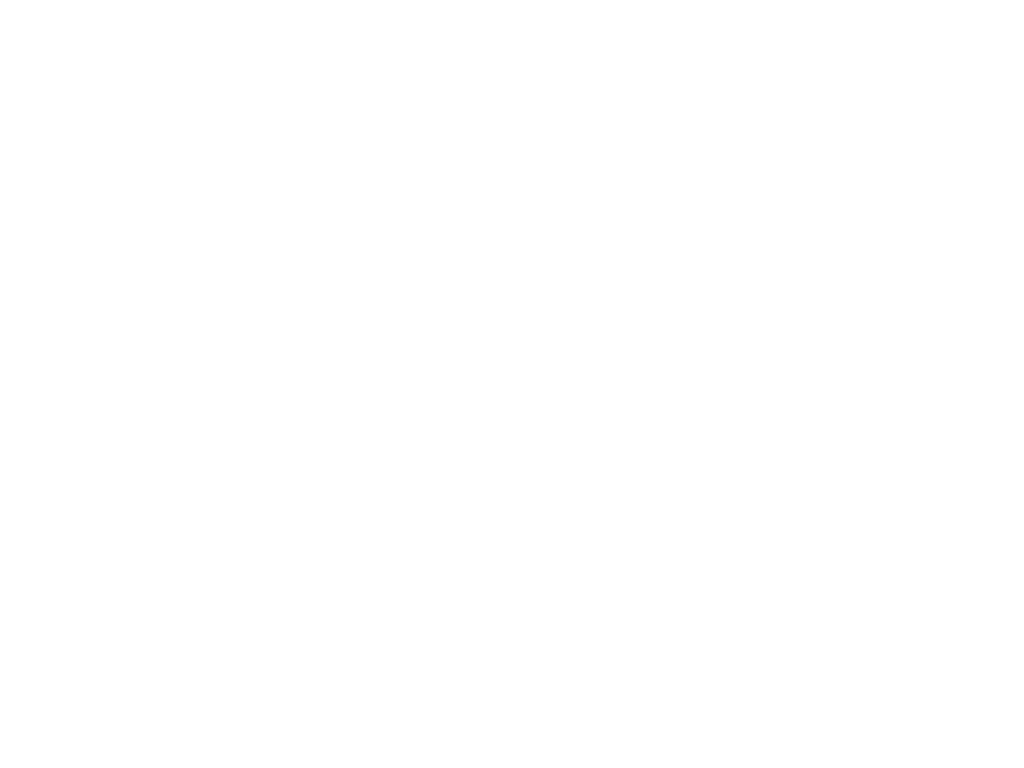
4.1 Feature Association

DataRobot’s Feature Association Matrix is populated by default by features from DataRobot’s Informative Features feature list. The Feature Association Matrix provides information on association strength between pairs of numeric and categorical features that are visually denoted by the opacity of the color (that is, num/cat, num/num, cat/cat, where lighter shades indicate weaker association and vice versa) and feature clusters. Clusters, families of features denoted by color on the matrix, are features partitioned into groups based on their association structure.

Some of the noted benefits of the Feature Association Matrix include:

* Understand the strength and nature of associations within the data;
* Detect families of pairwise association clusters; and,
* Identify clusters of high-association features prior to model building.

The Feature Association Matrix lists up to the top 50 features, selected by Importance Score, on both the X and Y axes, where the intersection of a feature pair provides an indication of their level of association. By default, the matrix displays by the Mutual Information values and sorts by the cluster.



The following are some general takeaways from looking at the matrix above:

* Each dot represents the association between two features (a feature pair), where the opacity of the color denotes the pair-wise strength of association.
* Each cluster is represented by a different color.
* The opacity of color indicates the level of association 0 to 1, between the feature pair. Levels are measured by the set metric, either mutual information or Cramer’s V.
* Shaded gray dots indicate that the two features, while showing some association, are not in the same cluster.
* White dots represent features that were not categorized into a cluster.
* The target feature, if present, is indicated by two small concentric circles next to the feature name.

4.2 Data Source Overview and Appropriateness

Explain how the data is suitable and relevant for the business problem and model use. For example:

Describe how, and from where, the data was obtained.

Provide a detailed description of the data source and its relevance to the business problem being addressed by this model.

Assess whether the data used for model development is appropriate given the populations to which the model will be applied.

If the model development and model implementation data sources differ, provide a detailed explanation justifying the use of different data sources.

4.3 Input Data Extraction, Preparation, and Quality & Completeness

Provide a detailed description of the data extraction and preparation process, and discuss any analysis conducted to confirm the data are complete and of sufficient quality (e.g., data validation). Include a detailed description of the data extraction process, hierarchical by extraction and preparation stage, and calling sequence. Provide data extraction code (e.g., SQL, Spark, etc.) in the Appendix.

Review and comment on any data weaknesses and limitations and their probable potential effects on the model. For example, data truncation, extraction timing, through-the-cycle data, and data exclusions could potentially cause unintended effects on the model.

4.4 Data Assumptions

Comment on data assumptions, the potential effects on the model, and any mitigating data controls. For example, assumptions related to data truncation, extraction timing, through-the-cycle data, reliability of source system, manual data overrides or imputation, and data exclusions could potentially cause unintended effects on the model.

5 Model Theoretical Framework and Methodology

5.1 Model Development Overview

DataRobot simplifies model development by performing a parallel heuristic search for the best model or ensemble of models, based on both the characteristics of the data and the prediction target. While some machine learning techniques tend to consistently outperform others, it is rarely possible to say in advance which will perform best for a given business problem. Therefore, during the modeling process, DataRobot develops dozens of independent challenger models, exposes the details of how these models were built and how they perform, and enables the user to select the best model for the particular business problem being addressed.

The fundamental workflow within DataRobot for model development is as follows:

* Rapid Data Ingestion: User creates a modeling dataset that includes the prediction target and loads into DataRobot
* Target Selection: User selects the prediction target; DataRobot detects whether the target is categorical or continuous. If the target is categorical, DataRobot selects and builds classification blueprints. If the target is continuous, DataRobot selects and builds regression blueprints. DataRobot also selects an optimization performance metric based on the type of supervised learning problem, which can also be changed by the user
* Automated Data Preparation: DataRobot analyzes the input data and automatically performs advanced preprocessing steps that are discussed in detail in this document. DataRobot also automatically partitions the input dataset into learning, validation and holdout dataset; these can also be defined by the user.
* DataRobot uses information about the selected target variable and predictors to define a set of candidate blueprints for analysis. It then trains models for each blueprint and ranks them on the model Leaderboard based on an out-of-sample validation accuracy score.
* Transparent Model Evaluation and Selection: DataRobot has built-in diagnostic tools to assess model accuracy and performance. Once DataRobot has trained and tested models, users can access them from the Leaderboard. From there, users can review model accuracy and, using built-in model diagnostic tools, understand how each independently built model performs. DataRobot provides many metrics for evaluating model accuracy, such as AUC, Log-Loss and RMSE. DataRobot's Leaderboard actively tracks performance of candidate models using out-of-sample data for comparison purposes.
* Model Deployment and Monitoring: Once the final model is selected, DataRobot provides efficient solutions for deployment (i.e., model implementation) and monitoring. These features enable the model owner to effectively manage model controls in accordance with Model Risk Management standards and policies.

5.2 Model Assumptions

This section should include model limitations, potential effects, and any mitigating controls in place. Limitations come in part from weaknesses in the model due to its various shortcomings, approximations, and uncertainties. Regulatory guidance refers to limitations as "...a consequence of assumptions underlying a model that may restrict the scope to a limited set of specific circumstances and situations." This section should include model limitations, potential effects, and any mitigating controls in place. Also include details here about the implementation of the models, what data will be used for scoring and why it is reasonable to think that the training data and the scoring data will be similar.

Machine learning methods can produce more accurate predictive models than traditional statistical regression methods because they are more flexible and rely less on statistical assumptions. For instance, ordinary least squares regression requires that the Gauss Markov assumptions are supported, which ensures that the model is unbiased and efficient.

Traditional statistical regression techniques rely on formal hypothesis testing for variable significance and feature selection (e.g., t-test, p-value, standard error). These hypothesis tests tend to have distributional and independence assumptions that may not be supported by the data. Machine learning methods, on the other hand, offer more flexibility in defining the model structure, which typically results in better model performance. Because machine learning includes methods that do not rely on formal hypothesis testing to demonstrate model validity, and because heuristic-style feature selection methods (e.g., stepwise selection) are not used in most machine learning approaches, no such distributional assumptions are required. In this case, the only assumption being made is that the model training data is representative of the future scoring data. Of course, these assumptions must be closely monitored and tracked by the model's ongoing performance monitoring process.

A common limitation of machine learning methods is the potential for overfitting. Overfitting occurs when the model is trained too closely to the underlying training data and does not perform well out-of-sample. DataRobot utilizes a robust cross-validation and holdout methodology to ensure model performance is sound, reducing the risk of overfitting.

5.3 Model Methodology

The modeling workflow consists of the following elements, which connect to create the blueprint:

* Train-Time Image Augmentation
* Pretrained MobileNetV3-Small-Pruned Multi-Level Global Average Pooling Image Featurizer
* Robust Standardization
* Keras Slim Residual Neural Network Classifier using Training Schedule (1 Layer: 64 Units)

The following subsections include details for each node of the modeling blueprint.

5.3.1 Adds new images to the dataset by transforming copies of existing images.

This can boost predictions scores by helping to train a more generalizable model which has been exposed to more data. This technique is especially helpful for small datasets, where there is a higher risk of overfitting to the training data. As a tradeoff for seeing more data, this means that the models trained with augmentation will take longer. In addition, depending on the data set, some transformations can hurt prediction scores.

See “https://docs.datarobot.com/en/docs/modeling/special-workflows/visual-ai/tti-augment/index.html” documentation in DataRobot application for more information.

5.3.2 Image Pretrained CNN Featurizer

IMGFEA converts images to numeric feature vectors using a pretrained convolutional neural network.

This method used here is called transfer learning, where a model developed for a task is reused as the starting point for a model on a second task. The method is particularly useful when used on datasets where the sample size is small and when there isn’t enough variation in the target dataset to fully represent the target environment. DataRobot’s technique extracts features from various levels of the convolutional neural network initialized with pretrained weights that include low level details, like edges and corners all the way to very high level details, like human faces and vehicles, that generalize well to a variety of different problems and user cases.

The feature vectors produced (one to four, based on the parameters use\_low\_level\_features, use\_medium\_level\_features, use\_high\_level\_features, and use\_highest\_level\_features of this task) represent extracted characteristics of the image at different levels of complexity. These feature vectors are then used downstream as input to a modeler.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Name | Description | Best Searched |
| select | Featurizer Pooling Type | Type of summarizer to use to squash the multi dimensional CNN features applied on initial, intermediate, and top convolutional layers of the network. values: ['avg', 'gem', 'max', 'gem-max', 'avg-max'] | None |
| select | Pretrained model | Pretrained model architecture. DarkNet:This simple neural network consists of eight 3x3 convolutional blocks with batch normalization, Leaky ReLu activation, and pooling. The channel depth increases by a factor of two each block. The network has nine layers in total, including a final dense layer. https://pjreddie.com/darknet/ DarkNet-Pruned:This simple neural network consists of eight 3x3 convolutional blocks with batch normalization, Leaky ReLu activation, and pooling. The channel depth increases by a factor of two each block. The network has nine layers in total, including a final dense layer.. https://pjreddie.com/darknet/ This is the same as DarkNet but with BatchNormalization layers removed after the Conv2D operation. EfficientNet-b0:EfficientNet-b0 is the fastest network in the EfficientNet network family, which were developed in 2019 using a “multi-objective neural architecture search that optimizes both accuracy and FLOPS.” The b0 model notably outperforms ResNet-50 top-1 and top-5 accuracy on ImageNet while having ~5x fewer parameters (ResNet has 26M while b0 has 5.3M). The main building of the EfficientNet models is the mobile inverted residual bottleneck (MBConv) convolutional block &mdash; which serves as a bottleneck, similar to the fire module in SqueezeNet, to constrain the number of parameters, but is “inverted” because the shape is opposite (in the mobile inverted residual bottleneck, the layer order is expand then squeeze). https://arxiv.org/abs/1905.11946 EfficientNetV2-S-Pruned:EfficientNetV2-S-Pruned is the latest Neural Network part of EfficientNets family. Combining all previous insights from EfficientNetV1 models (2019), and applying newly found Fused-MBConv approach by Google Neural Architecture Search: 1. Replaces “DepthwiseConv2D 3x3 followed by Conv2D 1x1” with “Conv2D 3x3” (operation is called Fused-MBConv). 2. Improves training procedures. Models are now pre-trained with over 13M+ images from 21k+ classes. In addition, DataRobot applies a layer-reducing “pruning operation”, where Conv2D+BatchNorm can be merged into a single linear transformation, keeping only the Conv2D layer. This reduces the total number of layers by 110, while preserving the same accuracy. See “EfficientNetV2: Smaller Models and Faster Training” (https://arxiv.org/pdf/2104.00298) for more information. https://arxiv.org/pdf/2104.00298 EfficientNet-b0-Pruned:The pruned architecture of EfficientNet-b0 version, meaning layer reduction. Conv2D+BatchNorm and DepthWise2D+BatchNorm can be merged into a single linear transformation, keeping only Conv2D/DepthWise and adding the value of BatchNorm as bias to the convolution. This results in an architecture with ~50 less layers and a significantly faster inference stage. EfficientNet-b4:The neural network in DataRobot likely to be the most accurate for a given dataset. The main goal in developing the EfficientNet family of models is to create a simple way to “scale up” the size of a model to obtain more accuracy at the cost of more computation. The implementation of the b4 model scales up (from Efficient-b0) the width of the network (number of channels in each convolution) by 1.4, and the depth of the network (the number of convolutional blocks) by 1.8, providing a more accurate and slower model than b0 but with results comparable to ResNext-101 or PolyNet. https://arxiv.org/abs/1905.11946 EfficientNet-b4-Pruned:The pruned architecture of EfficientNet-b4 version. Conv2D+BatchNorm and DepthWise2D+BatchNorm can be merged into a single linear transformation, keeping only Conv2D/DepthWise and adding the value of BatchNorm as bias to the convolution. This results in an architecture with ~100 less layers and a significantly faster inference stage. MobileNetV3-Small-Pruned:Mobilenet V3 is the latest incarnation of the MobileNet family of neural networks, which are specially designed for mobile phone CPUs and other low-resource devices. It comes in two variants: MobileNet3-Large and MobileNet3-Small. MobileNetV3-Large targets high resource usage, whereas MobileNet3-Small is for low resource usage. MobileNetV3-Small is 6.6% more accurate the previous MobileNetV2 network with almost identical or better latency. In addition to these lightweight blocks and operations, pruning is applied for even faster feature extraction. This pruned version keeps the same architecture but with signigicantly lower number of layers ( ~50 ). Conv2D + BatchNormalization and Depthwise2D + BatchNormalization are merged into single Conv2D layer. “Searching for MobileNetV3” https://arxiv.org/abs/1905.02244 PreResNet10:PreResNet10 is based on ResNet, except within each residual block the batch norm and ReLu activation happen \_before\_ rather than \_after\_ the convolutional layer. This implementation of the PreResNet architecture has four PreRes blocks with two convolutional blocks each, which yield 10 total layers, including an input convolutional layer and output dense layer. According to the paper (p.14), the model’s computational complexity scales linearly with the depth of the network, so this model is about 5x faster than ResNet50. However, because the richness of the features generated by the featurizer can affect the fitting time of downstream modelers like XGB with Early Stopping, the time taken to train a model using a deeper featurizer like ResNet50 could be even more than 5x. https://arxiv.org/abs/1603.05027 ResNet50:This classic neural network is based on residual blocks containing skip-ahead layers, which allow for very deep networks that still train effectively. In each residual block, the inputs to the block are run through a 3x3 convolution, batch norm, and ReLu activation - twice. Then that result is added to the inputs to the block, which effectively turns that result into a residual of the layer. This implementation of ResNet has a total of 50 layers, including an input convolutional layer, 48 residual blocks, and a final dense layer. https://arxiv.org/abs/1512.03385 ResNet50-Pruned:This classic neural network is based on residual blocks containing skip-ahead layers, which allow for very deep networks that still train effectively. In each residual block, the inputs to the block are run through a 3x3 convolution, batch norm, and ReLu activation - twice. Then that result is added to the inputs to the block, which effectively turns that result into a residual of the layer. This implementation of ResNet has a total of 50 layers, including an input convolutional layer, 48 residual blocks, and a final dense layer. The only difference between ResNet50-Pruned is that all BatchNormalization layers after Conv2D operation are removed, resulting in a network with ~50 less layers compared to original architecture. This makes it way faster for both CPU and GPU inference. https://arxiv.org/abs/1512.03385 SqueezeNet:The fastest neural network in DataRobot&mdash;this network is designed to achieve the speed of AlexNet with 50x fewer parameters, allowing for faster training, prediction , and storage size. It is based around the concept of fire modules, consisting of a combination of “squeeze” layers followed by “expand” layers, which aim to dramatically reduce the number of parameters used while preserving relatively high accuracy. This implementation of SqueezNet v1.1 has an input convolutional layer followed by eight fire modules of three convolutions each, leading to a total of 25 total layers. Original paper: https://arxiv.org/abs/1602.07360 Explanation of the differences introduced in v1.1: https://github.com/forresti/SqueezeNet/tree/master/SqueezeNet\_v1.1 Xception:This neural network is an improvement in accuracy over the popular Inception V3 network , which has comparable speed to ResNet-50 but with better accuracy on some datasets. The core idea of this network is that it saves on parameters by learning spatial correlations separately from cross-channel correlations. The core building block of this network is the depthwise separable convolution (a depthwise convolution + pointwise convolution) with residual layers added (similar to PreResNet-10). This building block aims to “decouple” the learning happening across the spatial dimensions (height and width) with the learning happening across the channel dimensions (depth), so that they are handled in separate parameters where interaction can be learned from other parameters downstream in the network. This network has a total of 36 convolutional layers, including has 11 convolutional layers in the “entry flow” where the width and height are reduced and the depth increases, and 24 convolutional layers where the size remains constant. https://arxiv.org/abs/1610.02357 Below is an estimate of the architectures grouped into “speed classes”. Non-Pruned Versions: 1st (Fastest): SqueezeNet, PreResNet-10, DarkNet 2nd: EfficientNet-b0 3rd: ResNet50 4th: ResNet50 5th: Xception 6th: (Slowest): EfficientNet-b4 Pruned Versions: 1st (Fastest): MobileneNetV3-Small-Pruned 2nd DarkNet-Pruned 3rd: EfficientNet-b0-Pruned 4th: EfficientNet-B4-Pruned & EfficientNetV2-S-Pruned 5th: (Slowest) ResNet50-Pruned We recommend using pruned version when available. Although ranking these neural network architectures on accuracy is difficult to do across all possible datasets (and input feature datatypes), note that speed is loosely inversely correlated to the accuracy. However, there are plenty of examples of datasets in which a faster model may be more accurate (i.e. squeezenet sometimes being more accurate than PreResNet, or EfficientNet-b0 often being more accurate than ResNet-50). It’s also worth noting that advanced tuning of combinations of hyperparameters (including neural network architectures) can yield unexpected interactions that can boost accuracy, as deep learning hyperparameter tuning is still an active area of research that develops as new research and new neural network architectures are published. For extra accuracy, use the EfficientNetV2-S-Pruned model, as it is the latest incarnation of DL research for Computer Vision from Google AI. (2021) values: [ 'darknet', 'darknet-pruned', 'efficientnet-b0', 'efficientnet-b0-pruned, 'efficientnet-b4', 'efficientnet-b4-pruned', 'efficientnetv2-s-pruned', 'mobilenetv3-small-pruned', 'preresnet10', 'resnet50', 'resnet50-pruned, 'squeezenet', 'xception', ] | None |
| select | automated\_feature\_reduction: | Decides whether or not to use ACE scores for feature reduction by removing the features that have very low ACE scores. values: ['True', 'False'] | None |
| int | batch\_size: | Number of images that will be featurized in a single batch. Larger values will lead to faster training/prediction times but will also increase resource consumption. Also, due to innate specifics of TensorFlow, any value larger than 1 may lead to instability of featurization results around the 1e-6 - 1e-7, which for some models can cause a prediction validation error. In those cases, it is recommended to retrain with batch\_size=1. values: [1, 4] | None |
| select | metric\_ace: | The type of metric to be used to compute univariate feature importance ACE | None |
| select | use\_high\_level\_features | Extracts features from one of the last convolution layers. This layer provides high level features that are more complex than low or medium level features, and typically combines patterns from various low and medium level features to form complex objects that entirely depend on the architecture chosen. Note that this feature level might have different meanings for different architectures&mdash;for shallow networksm this layer might still produce features that are considered by humans as low level features. This generally works for classification problems. values: [True, False] | True |
| select | use\_highest\_level\_features | Extracts features from the final hidden layer. This feature works well if the target problem contains “ordinary” images, such as animals, or vehicles, or foods, or plants. values: [True, False] | True |
| select | use\_low\_level\_features | Extracts features from one of the initial layer convolution layers. This layer provides low level visual features and will benefit problems that is vastly different from “ordinary” image datasets. values: [True, False] | False |
| select | use\_medium\_level\_features | Extracts features from one of the intermediate layer convolution layers. This layer provides medium level visual features that are more complex than low level features, and typically combines patterns from different levels of low level features. values: [True, False] | True |

5.3.3 Image Features Post Processor

Handles all the post processing in one task specifically for image based features from pretrained CNN networks.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Name | Description | Best Searched |
| select | pca\_whitening: | the type of pca whitening to use along with its input initialization type values: [`pca\_whitening`, `l2\_norm\_pca\_whitening`, `l1\_norm\_pca\_whitening`, `None`] | None |
| select | scaling\_method: | the scaling method to use before feeding it to further tasks values: [`Standardization`, `Robust\_Standardization`, `L1\_Normalization`, `L2\_Normalization`, `None`] | None |

5.3.4 Keras Neural Network Multi Classifier

Neural networks are a family of models inspired by biological neural networks (the central nervous systems of animals, in particular, the brain) and are used to estimate or approximate functions that can depend on a large number of inputs that are generally unknown. Neural networks are generally presented as systems of interconnected “neurons” which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning.

A neural network with no hidden layers is equivalent to a logistic or linear regression model depending on the activation function used (e.g., sigmoid vs. linear activation). Adding a “hidden layer”, a set of neurons followed by an activation, (between the input and output layer) into the neural network, introduces non-linearity. This allows the model to learn non-linear relationships between features, which can lead to significantly more powerful models than simple linear models.

Neural networks learn using an optimizer and back-propagation. In other words, repeatedly take a small batch of data, calculate the difference between predictions and actuals, and adjust the weights by a small amount, layer by layer, to generate predictions nearer to the actuals.

This form of modeling is very flexible allowing composition of arbitrary functions, but it is also much more sensitive to the input data than regular regression models, requiring special techniques such as batch normalization. Definitely consider leveraging neural networks when a task involves finding interactions within text data.

Keras is a high-level library for building neural networks using the Tensorflow framework for deep learning models. Keras provides flexibility for rapidly incorporating state-of-the-art deep learning models into DataRobot. Keras also supports sparse data, which can be particularly important for text-heavy data or categorical data with many levels.

This class is capable of producing standard neural network models with multiple hidden layers, as well as more advanced architectures, such as Self-Normalizing Neural Networks (as described in https://arxiv.org/abs/1706.02515) and Residual Connections (as described in https://arxiv.org/abs/1712.09913)

Self-Normalizing Neural Networks use very specific input initializers and a special activation function called “Scaled Exponential Linear Units” to prevent vanishing/exploding gradients without the use of batch normalization.

Residual networks include a direct connection from the inputs to the outputs, which smooths out the loss function and enables better optimization of the network.

SHAP: Although Keras models provide support for the use of SHAP for Feature Impact and Prediction Explanations, you may receive a warning indicating imprecise additivity, which suggests the results are misleading. This tends to happen more frequently for certain activation functions such as SELU and Swish.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Name | Description | Best Searched |
| int or select | batch\_size | The Keras neural networks are trained via SGD, in mini-batches. This parameter determines how many rows to consider for each mini-batch. Higher values will tend to train faster, but use more RAM. Setting too high can cause the model to have trouble converging, especially in more complicated models with more layers. Note: If this is set to ‘auto’, it will use a heuristic to calculate batch size based on the number of the rows in the dataset. Empirical testing found that larger datasets do better with larger batch sizes, and smaller datasets tend to do better with smaller batch sizes. By default, the auto heuristic will increase batch size by 1 for every 64 rows in the dataset, rounded up to the nearest power of 2. However, this will differ from dataset to dataset, as DataRobot will determine the best batch size heuristic on a dataset-by- dataset basis. Possible Values: {‘intgrid’: [1, 131072], ‘select’: [‘auto’]} | 1819 |
| int | double\_batch\_size | If 1, the batch size will be doubled every epoch (with the largest value being max\_batch\_size). Possible Values: [0, 1] | 0 |
| select | dropout\_type | Whether to use normal dropout or alpha dropout. Applies to both the hidden layers and the output layer. Use “normal” or “alpha”. Possible Values: [‘normal’, ‘alpha’] | normal |
| int | early\_stopping | The number of epochs with no improvement after which training will be stopped. If early\_stopping = 0, then there is no early stopping. If early\_stopping > 0, will check validation loss on the grid search test set, and terminate before hitting epochs if that loss increases early\_stopping consecutive times. Only one of stochastic\_weight\_average\_epochs and early\_stopping can be nonzero. Possible Values: [0, 1000] | 0 |
| int | epochs | Number of passes through the data to run. 1 epoch means the model will consider each point in the training data exactly once. Note that due to how stochastic gradient descent works, when loss is calculated to determine how to update the weights, they will be changed only by a small fraction (according to learning rate) of the difference between the guessed and actual target. This is why one will often choose to do multiple or even many passes through the data. Possible Values: [1, 1000] | 8 |
| select | hidden\_activation | Activation function to use for the hidden layers only. “relu” and “prelu” are usually good choices. Note that while units, hidden\_dropout, hidden\_batch\_norm, hidden\_l1, and hidden\_l2 are lists and can change layer-to-layer, hidden\_activation is the same for all hidden layers. Possible Values: [‘linear’, ‘sigmoid’, ‘hard\_sigmoid’, ‘relu’, ‘elu’, ‘selu’, ‘tanh’, ‘softmax’, ‘softplus’, ‘softsign’, ‘exponential’, ‘swish’, ‘mish’, ‘thresholdedrelu’, ‘leakyrelu’, ‘prelu’, ‘cloglog’, ‘probit’] | prelu |
| int | hidden\_batch\_norm | Whether or not to batch normalize each hidden layer. This can speed up model convergence. Be careful about setting hidden\_batch\_norm = 1 and hidden\_dropout > 0 at the same time. 1 = use batch norm, and 0 = do not use batch norm. Applies to all hidden layers. Possible Values: [0, 1] | 0 |
| select | hidden\_bias\_initializer | Initializer for the bias in the hidden layers. Used for all layers. Possible Values: [‘zeros’, ‘ones’, ‘random\_uniform’, ‘lecun\_uniform’, ‘glorot\_uniform’, ‘he\_uniform’, ‘random\_normal’, ‘lecun\_normal’, ‘glorot\_normal’, ‘he\_normal’, ‘truncated\_normal’, ‘VarianceScaling’, ‘orthogonal’] | zeros |
| float | hidden\_dropout | Fraction of activations to drop randomly on each forward pass of training, referred to as “dropout”. This regularizes the models and typically improves generalization. Float provided here will used to determine level of dropout of each layer. (Applies to all hidden layers.) Set to 0 for no dropout. Possible Values: [0, 0.99] | 0.05 |
| select | hidden\_initializer | Initializer for the hidden layer of the model. Recommended to leave at the default. Possible Values: [‘zeros’, ‘ones’, ‘random\_uniform’, ‘lecun\_uniform’, ‘glorot\_uniform’, ‘he\_uniform’, ‘random\_normal’, ‘lecun\_normal’, ‘glorot\_normal’, ‘he\_normal’, ‘truncated\_normal’, ‘VarianceScaling’, ‘orthogonal’] | he\_uniform |
| float | hidden\_l1 | L1 regularization to use for each hidden layer. Tends to select variables feeding into the hidden layer. 0 for no L1 regularization. This is a penalty coefficient that is applied to l1(weights) in the loss function. Applies to all hidden layers. Possible Values: [0, 1000000.0] | 0.0 |
| float | hidden\_l2 | L2 regularization to use for each hidden layer. Tends to shrink coefficients feeding into the hidden layer. 0 for no L2 regularization. This is a penalty coefficient that is applied to l2(weights) in the loss function. Applies to all hidden layers. Possible Values: [0, 1000000.0] | 0.0 |
| list of ints | hidden\_units | Number of units in the hidden layer of the network. If none, the model is equivalent to a simple regression model, fit via stochastic gradient descent (SGD), and will not find interactions between features. Specify a list of hidden units for multiple hidden layers, e.g. list(512, 256, 128) for 3 layers with decreasing numbers of units. Use “list()” to fit a model with no hidden layer. Possible Values: {‘length’: [0, 25], ‘int’: [1, 8192]} | [64] |
| int | hidden\_use\_bias | Whether or not to use a bias term for the hidden layers. Applies to all layers. 1 = use bias, and 0 = do not use bias. Applies to all hidden layers. Possible Values: [0, 1] | 1 |
| float | learning\_rate | Learning rate used for optimization. Lower learning rates can lead to more accurate models but require many more epochs to converge, and are more susceptible to local minima. If using a training schedule, learning\_rate represents the maximum learning rate. Possible Values: [1e-10, 1000] | 0.03 |
| select | loss | Loss function optimized by the model. Possible Values: [‘sparse\_categorical\_crossentropy’] | sparse\_categorical\_crossentropy |
| float | loss\_quantile\_level | Quantile level at which quantile loss should be calibrated. The default value of 0.5is equivalent to a median-optimized model. Only applicable for models using a quantile loss. Possible Values: [0.01, 0.99] | 0.5 |
| int | max\_batch\_size | The maximum batch size the model will consider, to avoid the doubling generating mini-batches that are too big. Applies only if double\_batch\_size is set to 1. Possible Values: [1048, 131072] | 131072 |
| select | optimizer | Which variant of SGD to use to fit the model. Recommended to use ‘adam’. Possible Values: [‘adam’, ‘sgd’, ‘rmsprop’, ‘adagrad’, ‘adadelta’, ‘adamax’, ‘nadam’, ‘adabound’] | adam |
| select | output\_activation | Activation for the final output layer of the network. Recommended to leave at the default. Possible Values: [‘linear’, ‘sigmoid’, ‘softsign’, ‘exponential’, ‘tanh’, ‘cloglog’, ‘softplus’, ‘probit’, ‘softmax’, ‘selu’, ‘elu’] | softmax |
| int | output\_batch\_norm | Whether or not to batch normalize the output layer. This can speed up model convergence. 1 = use batch norm, and 0 = do not use batch norm. Possible Values: [0, 1] | 1 |
| select | output\_bias\_initializer | Initializer for the bias in the hidden layers. Used for all layers. Set to “mean” to initialize the bias to the mean of the target, which can often help speed up convergence. Possible Values: [‘zeros’, ‘ones’, ‘random\_uniform’, ‘lecun\_uniform’, ‘glorot\_uniform’, ‘he\_uniform’, ‘random\_normal’, ‘lecun\_normal’, ‘glorot\_normal’, ‘he\_normal’, ‘truncated\_normal’, ‘VarianceScaling’, ‘orthogonal’] | mean |
| select | output\_initializer | Initializer for the final output layer of the network. Recommended to leave at the default. Possible Values: [‘zeros’, ‘ones’, ‘random\_uniform’, ‘lecun\_uniform’, ‘glorot\_uniform’, ‘he\_uniform’, ‘random\_normal’, ‘lecun\_normal’, ‘glorot\_normal’, ‘he\_normal’, ‘truncated\_normal’, ‘VarianceScaling’, ‘orthogonal’] | he\_normal |
| float | output\_l1 | L1 regularization to use for the output layer. Tends to select variables feeding into the output layer. 0 for no L1 regularization. This is a penalty coefficient that is applied to l1(weights) in the loss function. Possible Values: [0, 1000000.0] | 0.0 |
| float | output\_l2 | L2 regularization to use for the output layer. Tends to shrink coefficients feeding into the output layer. 0 for no L2 regularization. This is a penalty coefficient that is applied to l2(weights) in the loss function. Possible Values: [0, 1000000.0] | 0.0 |
| int | pass\_through\_inputs | If pass\_through\_inputs = 1, will add a connection of the inputs directly to the output layer. This added connection is often referred to as a skip connection. When a neural network leverages skip connections, it is referred to as a “residual neural network”. Possible Values: [0, 1] | 1 |
| int | prediction\_batch\_size | Batch size for predictions. Higher settings will use more RAM at prediction time, but will tend to be faster. This setting does not affect training at all, and it is recommended to leave it at the default. If a model is using too much RAM on a dedicated prediction server, you can try advanced tuning this parameter to a lower value, like 4096, 2048 or 1024. This will make batch predictions slower, but then they will use less RAM, without having any effect on the model’s accuracy. Also note that this parameter won’t affect 1-row-at-a-time predictions, which occur in batch sizes of 1. Possible Values: [1, 131072] | 8192 |
| int | random\_seed | Random seed to used to seed all operations which use a seed while constructing the network, such as dropout, initial weights, initial bias, etc. Possible Values: [0, 2147483646] | 42 |
| int | stochastic\_weight\_average\_epochs | Number of passes through the data over which to average network weights. If 0, then there is no SWA. If greater than 0, then SWA is applied over the final N=`stochastic\_weight\_average\_epochs` epochs. Must be <= epochs. Only one of stochastic\_weight\_average\_epochs and early\_stopping can be nonzero. Possible Values: [0, 1000] | 0 |
| string | training\_schedule\_curve | The function to use to transition between points in the training schedule. Generally, ‘exponential’ works well when the majority of the training schedule should be spent at low learning rates, linear works well when the schedule should spend equal time at all learning rates, and cosine works well when more time should be spent at both high and low learning rates. Cosine is used by default due to providing a warm-up and warm-down effect, while spending more time at high learning rates. Possible Values: [‘linear’, ‘exponential’, ‘cosine’] | cosine |
| int | training\_schedule\_cycle\_count | The number of cycles in the training schedule, before scaling down the learning rate even further in order to minimize the loss function. Possible Values: [0, 1000] | 1 |
| float | training\_schedule\_cycle\_scale | Defines the scale of each cycle in the training schedule. The maximum learning rate used in the schedule is defined by learning\_rate. The minimum learning rate used in each cycle is defined by training\_schedule\_cycle\_scale \* learning\_rate. Each cycle will start at the minimum learning rate, increase to the maximum learning rate, and fall back to the minimum learning rate. Possible Values: [0.0, 1.0] | 0.04 |
| float | training\_schedule\_cycle\_warm\_up\_fraction | Defines what fraction of the cycle will be used to build up to the maximum learning rate. By default, the cycle will begin at the minimum learning rate and reach the maximum learning rate 25% of the way through the cycle, and then decrease for the next 75% to the minimum learning rate. Possible Values: [0.0, 1.0] | 0.25 |
| float | training\_schedule\_post\_cycle\_scale | Defines the scale of the training schedule during the “warm down” step. By tuning the learning rate down even further during the warm down step, we encourage the model to descend into sharp loss function minima. By default, this will be 0.2% of the learning\_rate, which we represent as a decimal: 0.002. Possible Values: [0.0, 1.0] | 0.002 |
| float | training\_schedule\_warm\_down\_fraction | Defines how large the fraction of all iterations should be dedicated to decreasing the learning rate after all cycles, in an effort to fine-tune the weights to ensure we descend as much as possible into the current minimum of the loss landscape. By default, this will be the last 25% of training. In other words, all cycles are completed in the first 75%. Possible Values: [0.0, 1.0] | 0.25 |
| int | use\_training\_schedule | Whether or not to use a training schedule. If this is enabled, the learning rate, and if the optimizer supports it momentum, are adjusted on a schedule. The learning rate will start small, increase (“warm-up”) to the given learning\_rate and decay (“warm down”) back to the minimum. This will repeat according to the training\_schedule\_cycle\_count, until only the warm\_down\_fraction of iterations remain, when it will decay the learning rate even further for the remainder of training. Possible Values: [0, 1] | 1 |

5.4 Literature Review and References

* Pan, S. J., and Yang, Q. “A Survey on Transfer Learning” Knowledge and Data Engineering, IEEE Transactionson 22 (10): 1345-1359. https://www.cse.ust.hk/~qyang/Docs/2009/tkde\_transfer\_learning.pdf
* Bishop, Christopher M. “Neural networks for pattern recognition.” Oxford university press, 1995. https://www.microsoft.com/en-us/research/people/cmbishop/prml-book
* Dahl, George E., Tara N. Sainath, and Geoffrey E. Hinton. “Improving deep neural networks for LVCSR using rectified linear units and dropout.” Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013. http://www.csri.utoronto.ca/~hinton/absps/georgerectified.pdf
* Hinton, Geoffrey E., et al. “Improving neural networks by preventing co-adaptation of feature detectors.” arXiv preprint arXiv:1207.0580 (2012). https://arxiv.org/abs/1207.0580
* Sergey Ioffe, Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.” JMLR: Workshop and Conference Proceedings. No. 32. 2015. http://jmlr.org/proceedings/papers/v37/ioffe15.pdf
* Diederik Kingma, Jimmy Ba. “Adam: A Method for Stochastic Optimization.” arXiv preprint arXiv:1412.6980 (2015) https://arxiv.org/abs/1412.6980
* Martin Abadi, et al. “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.” http://download.tensorflow.org/paper/whitepaper2015.pdf
* Quoc V. Le, et al. “On optimization methods for deep learning.” Paper presented at the meeting of the ICML, 2011. http://ai.stanford.edu/~quocle/LeNgiCoaLahProNg11.pdf
* Gunter Klambauer, et al. “Self-Normalizing Neural Networks” https://arxiv.org/abs/1706.02515
* Hao Li “Visualizing the Loss Landscape of Neural Nets” https://arxiv.org/abs/1712.09913
* Leslie N. Smith, Nicholay Topin “Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates” https://arxiv.org/abs/1708.07120
* Leslie N. Smith “A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay” https://arxiv.org/abs/1803.09820

5.5 Alternative Model Frameworks and Theories Considered

As stated by regulatory guidance, comparison with alternative theories and approaches provides guidance for final model selection and is a fundamental component of a sound modeling process.

DataRobot develops dozens of alternative models, exposes the details of how these models were built and how they perform, and enables the user to select the best model for the particular business problem being addressed.

During the model development process, DataRobot considered the following alternative models. The final model was selected based on model performance as well as an analysis of model diagnostics and expert business judgment.

The performance metric used for this project was LogLoss. The model types considered during the model selection process included the following models, which are sorted by the Holdout score.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model Name | Validation Score | Cross Validation Score | Holdout Score | Sample Percentage |
| Regularized Logistic Regression (L2) | 0.0509 | N/A | 0.0482 | 100.0 |
| Baseline Image Classifier | 1.5759 | 1.6307 | 1.5527 | 71.9746 |

5.6 Variable Selection

The model's variable selection process includes a balance of quantitative analysis and key domain knowledge about the underlying business problem (i.e., expert judgment). The subsections below describe:

* DataRobot Quantitative Analysis: Key components related to variable selection that are automated by DataRobot.
* Expert Judgment and Variable Selection: Summary of the expert judgment used during the variable selection process.
* Final Model Variables: Final feature list chosen.

5.6.1 DataRobot Quantitative Analysis

A feature list is a defined set of features (variables) that DataRobot can use for modeling. DataRobot automatically creates three feature lists (described below) for each project. Users, however, can create customized feature lists that contain a subset of the total feature set, and use the new list to train new, alternative models. The default feature lists are described below:

* Informative Features (default): Features that pass a "reasonableness" check that determines whether they contain useful information. For example, DataRobot excludes features it determines are low information, such as a column containing all ones, duplicate columns, or a feature with too few values. The Informative Features list is sorted by each feature's correlation with the target variable.
* Raw Features: All features (variables) in the dataset, including those excluded from the Informative Features list.
* Univariate Selection: Features that meet a certain threshold for non-linear correlation with the selected target. DataRobot calculates, for each entry in the Informative Features list, the feature's individual relationship against the target.

Users also have the option to create user-defined feature transformations, which can then be included in a feature list for model exploration and to determine relative feature importance. Importance is measured using the information content of the variable; the calculation is done independently for each feature in the dataset. Features are then ranked on the Project Data from most to least important. This score represents a measure of predictive power using only that variable to predict the target. The score is measured using the project's accuracy metric that is defined by either the user (e.g., LogLoss) or the default assigned by DataRobot.

5.6.2 Expert Judgement and Variable Selection

This section should include additional detail regarding the variable selection process and any expert judgment used during feature selection.

5.6.3 Final Model Variables

Below is a list of the final set of model feature/independent variables, as well as summary statistics for the Keras Slim Residual Neural Network Classifier using Training Schedule (1 Layer: 64 Units) model.

The Model Features and Summary Statistics table provides a brief overview of the summary statistics of model features. This includes Feature Name, variable type (Var Type), number of unique values (Unique), Number of missing values (Missing), Mean, Standard Deviation (Std Dev), Median, Minimum Value (Min), Maximum Value (Max) and Assessment of target leakage risk (Target Leakage).

5.6.3.1 Model Features and Summary Statistics

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Feature Name | Var Type | Unique | Missing | Mean | Std Dev | Median | Min | Max | Target Leakage |
| ID | Image | 1413 | 0 | N/A | N/A | N/A | N/A | N/A | N/A |
| target | Numeric | 17 | 0 | 8.0 | 4.79 | 8.0 | 0.0 | 16.0 | N/A |

The last column in this table is an assessment of target leakage risk. DataRobot automatically tests for target leakage on a per-feature basis during the Autopilot process. Target leakage, sometimes called data leakage, occurs when a model is trained using a dataset that includes information that would not be available at the time of prediction. This can produce overly optimistic model performance results during training, given a feature will near-completely describe the target (e.g., the number of late payments on a loan as a predictor for loan default at loan application date.)

DataRobot tests for target leakage risk using Alternating Conditional Expectation (ACE) to measure the association between each feature and the target; the ACE score is normalized using the project optimization metric so that its value is in the range [0,1]. If above a certain threshold (see below), DataRobot will create a new feature list with those features flagged and possibly removed, and the user is notified by a banner in the user interface during modeling. Notably, because the definition of target leakage is directly tied with prediction time and not strength of association between a feature and the target, it's possible for DataRobot to not identify all sources of target leakage. Therefore, to reduce the risk for potential target leakage in the feature list, it's important to apply subject matter expertise.

The thresholds for target leakage risk are based on a normalized ACE score:

* High risk: > 0.975, flagged and removed
* Moderate risk: > 0.85, flagged but not removed
* Low risk: < 0.85, no action

6 Model Performance and Stability

6.1 Model Validation Stability

To find patterns in a dataset from which it can make predictions, an algorithm must first learn from a historical example – typically from a historical dataset that contains the output variable you want to predict. However, if a model is trained too closely on its training data then it may be overfit. Overfitting is a modeling error that occurs when a model is too closely fit to training data and therefore performs poorly on out-of-sample data (data that was not used to train the model). Overfitting generally results in an overly complex model that explains idiosyncrasies and random noise in the training data, rather than the underlying trends that the model was intended to capture. To avoid overfitting, the best practice is to evaluate model performance on out-of-sample data. If the model performs very well on in-sample data, (the training data) but poorly on out-of-sample data, that may be an indication that the model is overfit.

DataRobot uses standard modeling techniques to validate model performance and ensure that overfitting does not occur. DataRobot used a robust model k-fold cross-validation framework to test the out-of-sample stability of a model's performance. In addition to cross-validation partitioning, DataRobot uses a holdout sample to further test out-of-sample model performance and ensure the model is not overfit.

The following procedure was used during development to insure that overfitting did not occur:

* DataRobot set aside 10.0% of the training data as a holdout dataset. This dataset is used to verify that the final model performs well on data that has not been touched throughout the training process.
* For further model validation, the remainder of the data is divided into 5 cross validation partitions. To compensate for the overhead when working with large datasets, DataRobot first trains models on a smaller part of the data and uses only one cross-validation fold to evaluate model performance. Then, for the highest performing models, DataRobot increases the subset sizes. This results in only the best model being trained on the total cross-validation partition. For those models, DataRobot completes 5-fold cross-validation training and scoring. As a result, the mean score of complete model cross-validation is calculated across all folds. Those models that did not perform well will not have a cross-validation score. Instead, because they only had a "one-fold" validation, their score is reported in the Validation column.

The following figure summarizes the CV process used by DataRobot, where the blue denotes 90.0% of the data available for training, which is then divided into 5-folds for cross-validation and and red denotes the holdout sample.



DataRobot calculates the Cross Validation scores for each of the training data partitions or folds. The project metric used to calculate the score is LogLoss.

6.1.1 Cross Validation Scores

|  |  |
| --- | --- |
| Fold | Cross Validation Score (LogLoss) |
| Fold 1 | 0.39614 |
| Fold 2 | 0.48601 |
| Fold 3 | 0.36734 |
| Fold 4 | 0.50833 |
| Fold 5 | 0.49567 |

6.1.2 Data Partitioning Methodology

Data partitions were selected by means of random sampling.

6.2 Model Performance (Sample Scores)

As an additional layer of model validity, DataRobot not only evaluated the statistical metrics underlying the model, but also performed testing on out-of-sample records.

The performance metric used for this project was LogLoss. The model performance results are presented below for out-of-sample testing:

|  |  |
| --- | --- |
| Scoring Type | Score (LogLoss) |
| cross\_validation | 0.4507 |
| holdout | 0.5989 |
| validation | 0.3961 |

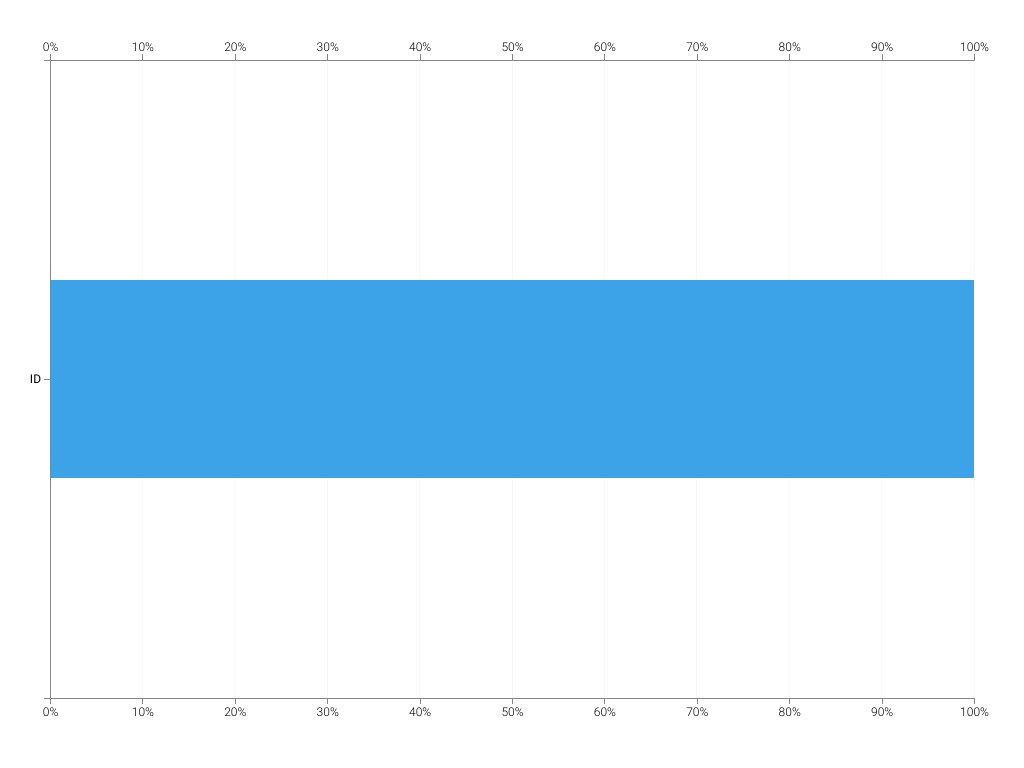
6.3 Sensitivity Testing and Analysis

6.3.1 Lift Chart

This section is not available when multiclass projects have more than 10 classes and are trained on 100% of the data.

6.3.2 Key Relationships

Feature Impact, which is available for all model types, works by altering input data and observing the effect on a models score. This technique is sometimes called Permutation Importance. The Feature Impact for a given column measures how much worse a models error score would be if DataRobot made predictions after randomly shuffling that column (while leaving other columns unchanged). DataRobot normalizes the scores so that the value of the most important feature column is first and the other subsequent features are normalized to it.



|  |  |  |
| --- | --- | --- |
| Feature Name | Impact Normalized | Impact Unnormalized |
| ID | 1.0 | 14.5902 |

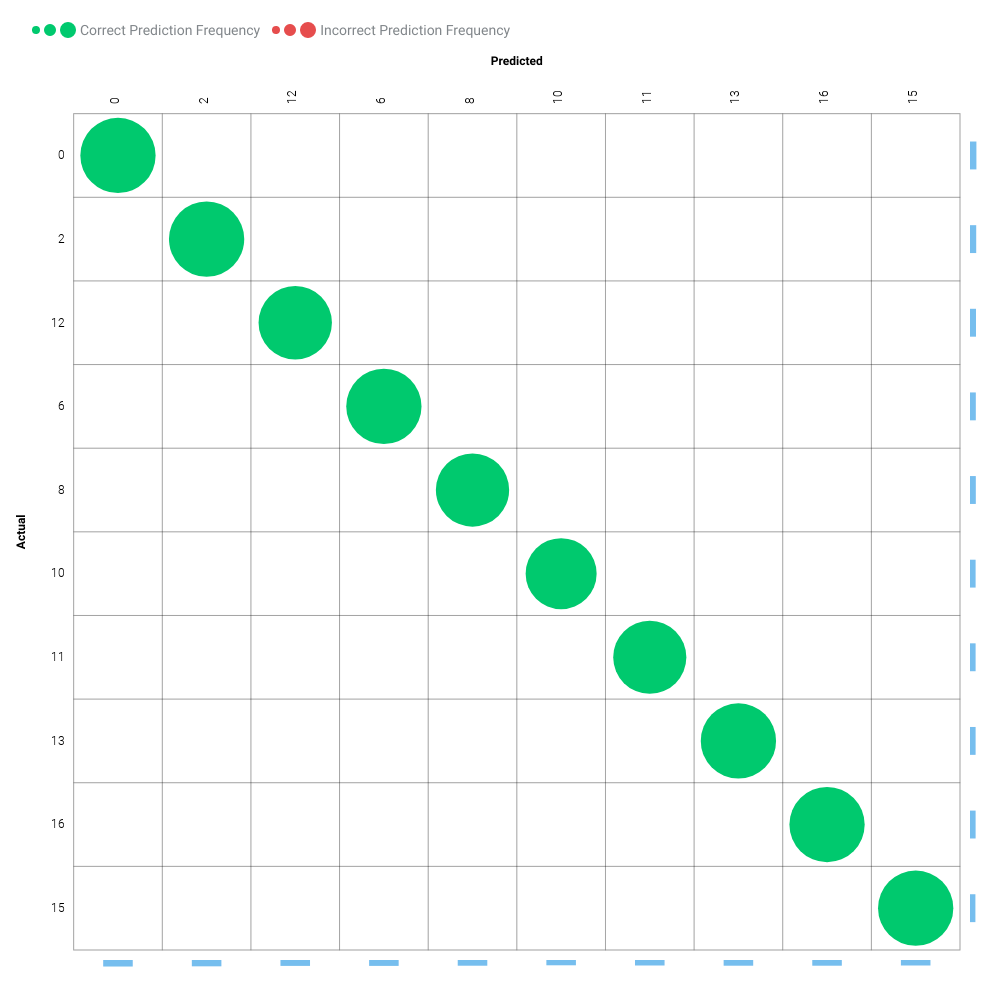
6.3.3 Accuracy

The table below lists, for each class, metrics to indicate how accurate the model was at predicting the class. The measurement is done by comparing the actual class assignment to the model's predicted assignment. Brief descriptions of each accuracy metric follow:

* F1 Score: A measure of the model's accuracy, based on the True Positive Rate and Positive Predictive Value.
* True Positive Rate: The ratio of true positives (correctly predicted as positive) to all actual positives. Also called sensitivity or recall.
* Positive Predictive Value: For all the positive predictions, the percentage of cases in which the model was correct. Also called precision.

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name | F1 Score | True Positive Rate | Positive Predictive Value |
| 0 | 1.0 | 1.0 | 1.0 |
| 1 | 0.9231 | 0.8571 | 1.0 |
| 10 | 1.0 | 1.0 | 1.0 |
| 11 | 0.9333 | 0.875 | 1.0 |
| 12 | 0.8889 | 0.8889 | 0.8889 |
| 13 | 0.8889 | 1.0 | 0.8 |
| 14 | 0.3636 | 0.3333 | 0.4 |
| 15 | 1.0 | 1.0 | 1.0 |
| 16 | 1.0 | 1.0 | 1.0 |
| 2 | 1.0 | 1.0 | 1.0 |
| 3 | 0.7692 | 0.7143 | 0.8333 |
| 4 | 0.5263 | 0.4545 | 0.625 |
| 5 | 1.0 | 1.0 | 1.0 |
| 6 | 1.0 | 1.0 | 1.0 |
| 7 | 0.4 | 0.5714 | 0.3077 |
| 8 | 1.0 | 1.0 | 1.0 |
| 9 | 1.0 | 1.0 | 1.0 |

For multiclass models, DataRobot provides a multiclass confusion matrix to help evaluate model performance. A confusion matrix is a visualization that helps you see which classes the model most often confuses with one another. The confusion matrix is the result of comparing the actual class against each of the possible predicted classes. The green points represent instances when the model predicted the correct class while the red points represent when the model was incorrect. The size of the points indicate the number of predictions made for a class. If your model has more than ten classes, only the first ten (based on actual frequency) are shown below, in descending order for the validation partition.



7 Model Implementation and Output Reporting

7.1 Version Control

DataRobot handles model and project version control automatically by tagging each model on the Leaderboard with a unique Model ID. The Model ID represents a single instance of a model type, feature list, sample size, and set of tuning parameter values. DataRobot also maintains unique Project IDs for each project, allowing accessibility to all models built for the project dataset. DataRobot's version control allows for reproducibility and traceability of the models it creates, which greatly increases the auditability of the model development process.

Users may also export Scoring Code for a DataRobot model in Java. You can download both a pre-compiled .jar file (with all dependencies included), plus the source code. Scoring Code is easy to deploy, test, and maintain on a variety of platforms, and you can inspect the generated Java code for complete transparency. DataRobot Scoring Code employs advanced features to ensure that predictions computed using generated Java code are the same as predictions computed inside DataRobot.