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 One-Hot Encoding Task
* 5.3.2 Median Value-Based Numeric Imputation (V2 with quick median algorithm)
* 5.3.3 Ridit transformer
* 5.3.4 Binning based on decision trees
* 5.3.5 Document-Term Matrix
* 5.3.6 Keras Neural Network Regressor
* 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
* 5.6.3.2 Data Quality Handling Report
* 6 Model Performance and Stability
* 6.1 Model Validation Stability
* 6.1.1 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 Sensitivity Analysis (Partial Dependence)
* 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.

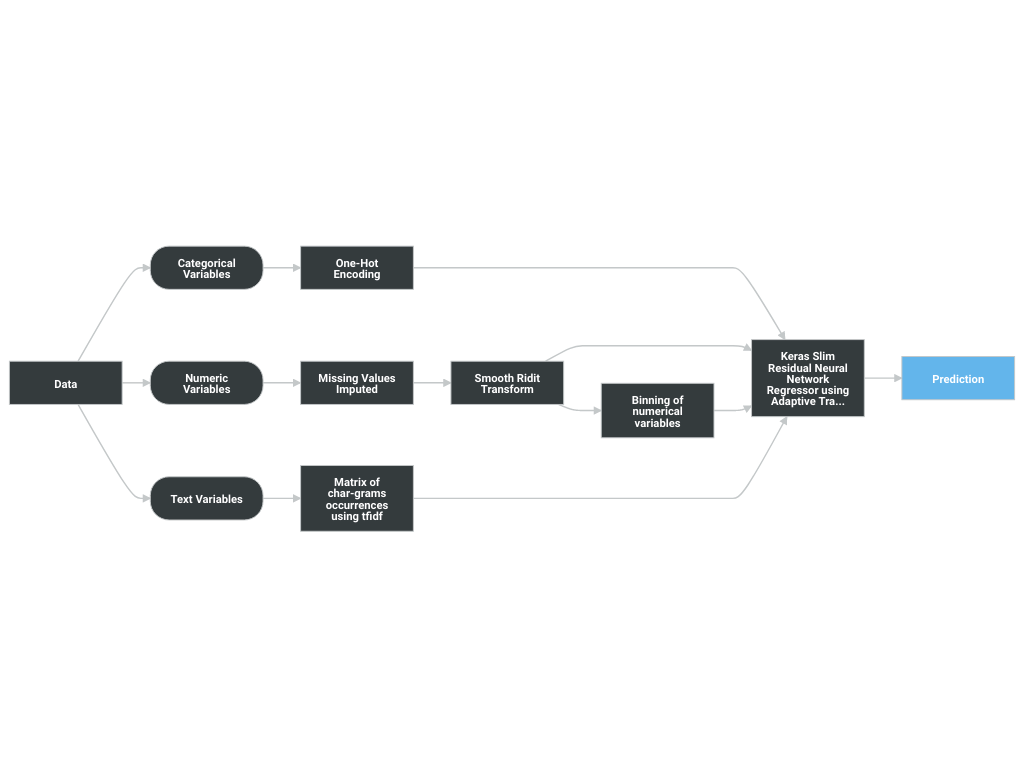
3.2 Model Development Purpose and Intended Use

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 Regressor using Adaptive Training Schedule (1 Layer: 64 Units). This model was developed in a project created with veeed602d97362576 of DataRobot. This model is denoted within DataRobot by the Project ID: 668eea805ded71e21fc89fd1 and the Model ID: 668fa96438f8ec8d44d2d0aa. The project was created on 2024-07-10 20:09:36.

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:

* One-Hot Encoding
* Missing Values Imputed
* Smooth Ridit Transform
* Binning of numerical variables
* Matrix of char-grams occurrences using tfidf
* Keras Slim Residual Neural Network Regressor using Adaptive 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 Gamma Deviance and the project included a total of 1,118,822 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 (Gamma Deviance) |
| holdout | 0.0057 |
| validation | 0.0057 |

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

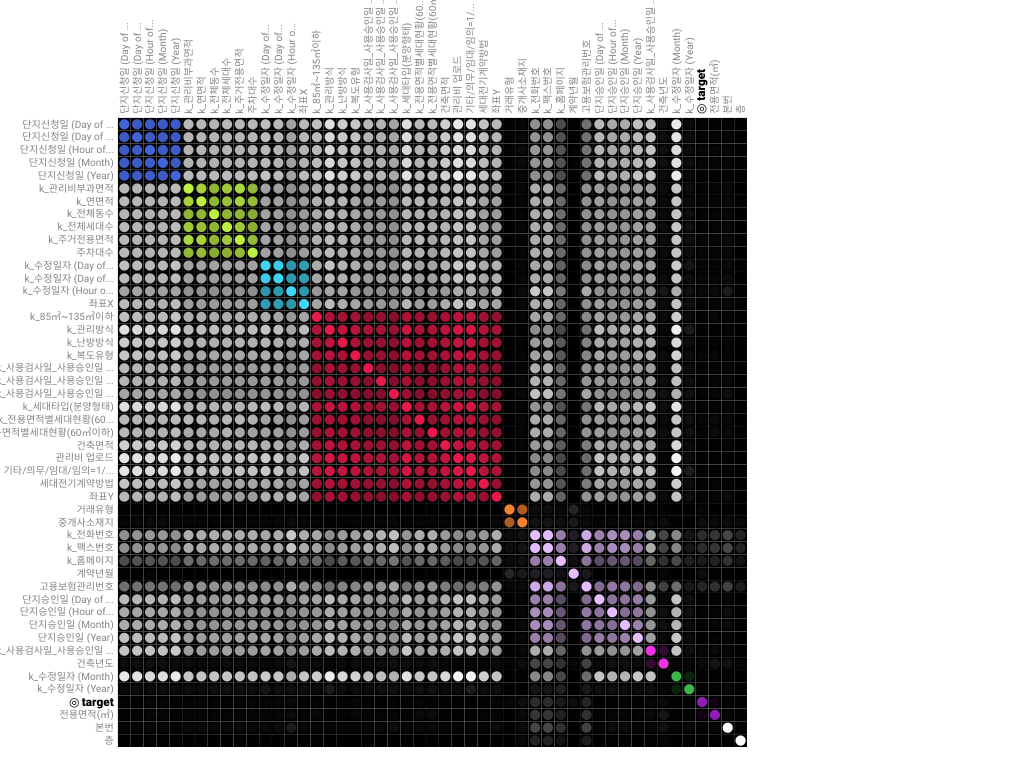
4.1 Feature Association

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:

* One-Hot Encoding
* Missing Values Imputed
* Smooth Ridit Transform
* Binning of numerical variables
* Matrix of char-grams occurrences using tfidf
* Keras Slim Residual Neural Network Regressor using Adaptive Training Schedule (1 Layer: 64 Units)

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

5.3.1 One-Hot Encoding Task

This transformer will do binary one-hot (aka one-of-K) coding. One boolean-valued feature is constructed for each of the possible string values that the feature can take. For inputs with only 2 unique values, only one boolean-valued feature will be constructed

This encoding is needed for feeding categorical data to many estimators, notably linear models and SVMs.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Name | Description | Best Searched |
| int | card\_max | An integer that specifies the maximum number of unique values. values: [1, 99999] | 50000 |
| int | card\_min | An integer that specifies the minimum number of unique values. values: [1, 99999] | 1 |
| bool | drop\_cols | drop\_cols, If True, drop last level of each feature values: [False, True] | False |
| select | flag | flag, If all, add highcat-cols to metadata values: ['None', 'all'] | None |
| int | max\_features | If the total number of categories created across all features exceeds this value, the top max\_features most frequent categories will persist. All others will be either thrown out or grouped. A value of None disables the limit. values: [1, 999999] | None |
| int | min\_support | The minimum number of records for a category to be represented in one hot encoding. If a category has fewer counts it will be grouped with other small cardinality values. values: [1, 99999] | 10 |

5.3.2 Median Value-Based Numeric Imputation (V2 with quick median algorithm)

For a numeric feature, impute rows of missing values with median value (V2).

Impute missing values on numeric variables with their median and create indicator variables to identify records that were imputed. A quick median algorithm (based on np.partition) is implemented to compute median feature value.

Imputation strategy:

A numeric feature is imputed with the median value if there are enough finite values in the feature samples used to train a numeric imputation task (e.g., > t, default: 50) and there are rows with NaN or infinite values in the samples to be imputed.

After imputation, the imputed numeric features will be scaled if the argument S is set to True. The feature will use scaled rounding (i.e., rounding to a logarithmic scale).

Imputation indicator:

The indicator column (0, 1) is added to indicate imputed rows if the numeric feature is imputed with : 1) the median value and with at least one row with NaN and 2) at least two unique values.

Example:

An imputation task is initialized with t=2.

Input numeric features of this task:

feature0, feature1, feature2, feature3

1, 2, np.nan, np.nan

2, 3, np.nan, 18

3, 2, np.nan, 16

4, 1, 13, 14

20, 1, 45, 46

Output numeric features of this task:

feature0, feature1, feature2, feature2-mi, feature3, feature3-mi

1, 2, 45, 1, 18, 1

2, 3, 45, 1, 18, 0

3, 2, 45, 1, 16, 0

4, 1, 13, 0, 14, 0

20, 1, 45, 0, 46, 0

In the imputation output, median value imputation is run on feature2 and feature3. The feature2-mi is the indicator column for the imputation on feature2. The feature3-mi is the indicator column for the imputation on feature3.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Name | Description | Best Searched |
| bool | scale\_small | True if small values (range of the numeric variable is <= 1) are to be scaled. values: [False, True] | False |
| int | threshold | Minimum number of required finite elements in a column to impute the data onto NaNs and INFs. values: [1, 99999] | 10 |

5.3.3 Ridit transformer

For a numeric feature, transform it to a ridit score based on percentile rank. The percentile score will be further adjusted to an interval between -1 and 1. The transformer can be configured to skip binary feature and date/time derived features. If the sparsity is higher than the sparsity\_threshold, data will be centered to the median and the output will be a sparse matrix.

The ridit transform is an extension of Bross’ (1958) RIDIT scoring method, which suggests the use of Ridit analysis for data that are ordered but not on an interval scale, such as injury categories. Bross’ (1958) RIDIT’s procedure is as follows: from a reference population with the same categories (of injury, for example), determine a “ridit” or score for each category. This category score is the percentile rank of an item in the reference population and is equal to the number of items in all lower categories plus one-half the number of items in the subject category, all divided by the population size. By definition, the mean of Bross ridit calculated for the reference population will always be 0.5.

The ridit transform extends the Bross ridit method by applying the method to numerical values and normalizing the score such that the mean calculated for the reference population will always be 0 and the score will be in the interval [-1,1].

Intuitively, the ridit transform can be interpreted to be an adjusted percentile score.

Ridit transform is not smooth and variable mapping is not continuous at the bin boundaries. DataRobot developed a “smooth” version of Ridit mapping, where the mapping inside of each bin is set to linearly increase to reach the other bin’s starting value. The middle point in the bin corresponds to the value of the original Ridit algorithm, and the mean of the distribution is still equal to 0.5. By using such approach, the mapping is continuous and predictions are consistent for the data values close to the boundary values.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Name | Description | Best Searched |
| bool | skip\_bins | If True, ridit transform will skip binary columns. values: [False,True] | False |
| bool | skip\_date\_features | If True, ridit transform will skip extracted features from the date column. values: [False,True] | False |
| float | sparsity\_threshold | If sparsity level is higher than the parameter, the matrix is converted to a sparse format. values: [0, 1] | 0.25 |

5.3.4 Binning based on decision trees

The binning task is a preprocessing method that transforms numerical variables into non-uniform bins.

It is a useful tool for building more powerful linear models that offers both good predictive accuracy and human understandable insights. It aims at automating the process of handcrafting binning strategies for data containing nonlinear relationships, which are difficult to capture using linear models.

The boundaries of the bins are determined by decision trees trained individually on each input feature. Hence, information about the target variable is utilized in the binning process. In particular, the threshold values used by the tree nodes are used as the bin boundaries. By default, an xgboost model is used to approximate the target variables that will be used by the decision trees.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Name | Description | Best Searched |
| int | max\_bins | Maximum number of resulting bins for each input feature values: [2, int(500)] | 256 |
| intgrid | minimum\_support | Minimum number of samples needed in each bin values: [1, int(1000)] | 50 |
| select | use\_xgb\_approximation | Whether to use xgboost model to approximate the target variable values: [0, 1] | True |
| floatgrid | xgb\_learning\_rate | Learning rate of the xgboost model values: [1e-5, 1] | 0.3 |
| select | xgb\_loss | Loss function to be optimized by the xgboost model values: ['binomial', 'ls', 'poisson', 'tweedie', 'gamma'] | ls |
| int | xgb\_n\_estimators | Number of estimators to use in the xgboost model values: [1, 20000] | 2000 |

5.3.5 Document-Term Matrix

A document-term matrix is a method for creating numeric features from a text column. These numeric features are generated based on the terms (i.e., tokens) present in the original text column.

There are various methods for determining how to create numeric features. In general, for a column containing text, DataRobot generates an additional column for each term in the original column. Then, for each row, the column value is based on whether the term was present in that row. In the binary (or occurrences) method, this value is either 1 (the term occurs in the row) or 0 (the term does not occur). In the count (or frequency) method, the column value is the count of the number of times the term appears in the original text row.

Several parameters affect the terms generated from a text feature. For example, the text can be separated by character (for example ‘cat’ will become three terms: ‘c’, ‘a’, and ‘t) or by word (for example ‘cat dog’ will become two terms: ‘cat’ and ‘dog’). Word or character n-grams can also be used, which creates terms based on groups of n consecutive words or characters (for example, 2-grams based on the characters of ‘cat’ would return the terms ‘ca’ and ‘at’).

You can exclude common terms that have little meaning. Words like ‘the’, ‘a’, and ‘is’ could be given large weight in a document-term matrix because of their high frequency, while not actually helping in creating predictive columns. These terms are commonly referred to as ‘stop\_words’ and can be removed from consideration when creating a matrix by setting stop\_words to True. You can also remove words based on their frequency: if they appear too often (more than max\_df) or not often enough (less than min\_df).

Another method of mitigating the impact of high-frequency, low-value terms is by reducing a term’s impact by weighting it. There are several weighting methods. A very common method is to use the tf-idf transform: term-frequency-inverse document-frequency. This method, weighs the frequency of a term in the text column by the inverse of the log of the total number of rows over how often the term appears across all rows. For example, if in a 10-row dataset the term appeared in a specific row 3 times and was in 2 of the total 10 rows, it would be given a value of 3 \* log(10 / 2) for that specific row in the associated column.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Name | Description | Best Searched |
| select | analyzer | When set to ‘word’, tokens are generated based on word n-grams. When set to ‘char’, character n-grams are used. values: ['word', 'char'] | char |
| bool | binary | When True, features are generated using the binary method: 1 if a term occurs and 0 if it does not. When False, features are based on the frequency of a term (how often the term is found in the document). values: [False, True] | True |
| select | language | Language to use in stemmers, lemmatizers (when available), and stop words. Note that not all languages support all preprocessing methods. English supports both stemming and stop words; Japanese does not support stemming. When the default word tokenizer is Japanese, it is strongly recommended that you leave all other parameters at their default values (in particular tokenizer, stop\_words, and stemmer) as mixing Japanese and English preprocessing will result in model errors. values: ['arabic', 'english', 'chinese', 'danish', 'dutch', 'finnish', 'french', 'german', 'hindi', 'hungarian', 'italian', 'norwegian', 'portuguese', 'romanian', 'russian', 'spanish', 'swedish', 'japanese', 'turkish', 'other'] | other |
| bool | lowercase | When True, convert all characters to lowercase before tokenizing. values: [False, True] | False |
| float or int | max\_df | Maximum document frequency. If a token is in more than this number of documents, it will not be included. For floating-point values, the parameter represents a proportion of documents. For integer values, the parameter represents absolute document counts. values: {'float': [0.0, 1.0], 'int': [1, 99999]} | 0.8 |
| multi | max\_features | Maximum number of features to generate. values: [1, 1e30] or None | 20000 |
| int | max\_ngram | Maximum value of n-grams to generate. values: [0, 99] | 3 |
| float or int | min\_df | Minimum document frequency. If a token is in less than this number of documents, it will not be included. For floating-point values, the parameter represents a proportion of documents. For integer values, the parameter represents absolute document counts. values: {'float': [0.0, 1.0], 'int': [1, 99999]} | 5 |
| int | min\_ngram | Minimum value of n-grams to generate. values: [0, 99] | 1 |
| select | norm | Normalization used in the rows of the matrix. values: [None, 'l1', 'l2'] | l1 |
| bool | smooth\_idf | Smooth idf adjusts idf by adding one to the document frequencies, as if an extra document was seen containing each term in the corpus. When use\_idf is set to False, this parameter has no effect. values: [False, True] | True |
| select | stemmer | NLTK word stemmer, which reduces words to their ‘stem’ so that they are not treated as separate words. For example, ‘running’ becomes ‘run’. Only considered when analyzer=’word’. Can stem English, and some other languages. Cannot stem Japanese. Highly recommended to set to None if language=’japanese’. Note that stemmers are not supported for GAM models, so do not change this preprocessing parameter from the default if the main model is a GAM or a GA2M. values: [None, 'snowball', 'lancaster', 'porter', 'wordnet'] | None |
| select | stop\_words | When True, stop words are removed in the vectorizer. DataRobot has several built-in, static lists of stop words; content is dependent on the language parameter. For example if stop\_words=True and language=”english”, DataRobot removes English stop words. If stop\_words=True and language=”french”, DataRobot removes French stop words. Note that not all languages support stop words. Also note that stop word removal often makes models less accurate and should be considered carefully. All of DataRobot’s text models are capable of dropping unneeded coefficients, which allows them to customize lists of stop words for the task at hand. This tuning often benefits from being able to analyze all words, even those that would be included on usual stop word lists. The wordcloud insight visualization has an option to hide stop words, which is usually preferable to removing them from the model completely. The language is specified in the language parameter. It is highly recommended to set to False if language=’japanese’. values: {'select': [False, True]} | False |
| bool | sublinear\_tf | When True, use sublinear term frequency (1 + log(tf)). values: [False, True] | False |
| select | tokenizer | NLTK word tokenizer, used to create terms for generating the matrix. Only considered when analyzer=’word’. If None, uses the default scikit-learn vectorizer, unless language=’japanese’, in which case ‘tiny-segmenter-jp’ is selected. See http://www.nltk.org/api/nltk.tokenize.html. values: [None, 'space', 'wordpunct', 'tweet', 'treebank', 'tiny-segmenter-jp', 'mecab'] | sklearn\_tokenizer |
| bool | use\_idf | When True, inverse document frequency (idf) is used. values: [False, True] | True |

5.3.6 Keras Neural Network Regressor

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’]} | 16384 |
| 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] | 3 |
| 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] | 3 |
| 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] | 0.0 |
| 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] | 0.015 |
| select | loss | Loss function optimized by the model. Possible Values: [‘mean\_squared\_error’, ‘mean\_absolute\_error’, ‘mean\_absolute\_percentage\_error’, ‘mean\_squared\_logarithmic\_error’, ‘squared\_hinge’, ‘hinge’, ‘logcosh’, ‘binary\_crossentropy’, ‘kullback\_leibler\_divergence’, ‘poisson’, ‘gamma’, ‘tweedie’, ‘quantile’, ‘cosine\_proximity’] | gamma |
| 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’, ‘selu’, ‘elu’] | exponential |
| 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] | 0 |
| 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\_uniform |
| 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

* Suits, Daniel B. “Use of dummy variables in regression equations.” Journal of the American Statistical Association 52.280 (1957): 548-551. http://www.jstor.org/stable/2281705?seq=1
* [1] Acuna, Edgar, and Caroline Rodriguez. “The treatment of missing values and its effect on classifier accuracy.” Classification, Clustering, and Data Mining Applications. Springer Berlin Heidelberg, 2004. 639-647. https://link.springer.com/chapter/10.1007/978-3-642-17103-1\_60
* [2] Feelders, Ad. “Handling missing data in trees: Surrogate splits or statistical imputation?” Principles of Data Mining and Knowledge Discovery. Springer Berlin Heidelberg, 1999. 329-334. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.36.7991&rep=rep1&type=pdf
* Bross, I. D. J. [1958]. “How to Use Ridit Analysis.” Biornetrics- 14, pg. 18-38. https://doi.org/10.2307/2527727
* Brockett, Patrick L., and Arnold Levine. “On a characterization of ridits.” The Annals of Statistics (1977): 1245-1248. https://projecteuclid.org/download/pdf\_1/euclid.aos/1176344010
* Luhn, Hans Peter. A statistical approach to mechanized encoding and searching of literary information. IBM Journal of research and development 1.4 (1957): 309-317.
* 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 Gamma Deviance. 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 |
| eXtreme Gradient Boosted Trees Regressor with Early Stopping (Gamma Loss) (Fast Feature Binning) | 0.0055 | N/A | 0.0056 | 64.0 |
| Light Gradient Boosted Trees Regressor with Early Stopping (Gamma Loss) | 0.0058 | N/A | 0.006 | 64.0 |
| Light Gradient Boosting on ElasticNet Predictions (Gamma Loss) | 0.0064 | N/A | 0.0064 | 64.0 |
| Keras Deep Residual Neural Network Regressor using Training Schedule (2 Layers: 512, 512 Units) | 0.0072 | N/A | 0.0073 | 64.0 |
| Keras Deep Residual Neural Network Regressor using Training Schedule (3 Layers: 512, 64, 64 Units) | 0.0075 | N/A | 0.0075 | 64.0 |
| Keras Wide Residual Neural Network Regressor using Training Schedule (1 Layer: 1536 Units) | 0.0079 | N/A | 0.0079 | 64.0 |
| Generalized Additive2 Model (Gamma Loss) | 0.0144 | N/A | 0.0143 | 64.0 |
| Keras Slim Residual Neural Network Regressor using Training Schedule (1 Layer: 64 Units) | 0.0166 | N/A | 0.0167 | 64.0 |
| Keras Deep Self-Normalizing Residual Neural Network Regressor using Training Schedule (3 Layers: 256, 128, 64 Units) | 0.0201 | N/A | 0.0202 | 64.0 |
| Keras Residual AutoInt Regressor using Training Schedule (2 Attention Layers with 2 Heads, 2 Layers: 96, 96 Units) | 0.0242 | N/A | 0.0242 | 64.0 |
| Elastic-Net Regressor (mixing alpha=0.5 / Gamma Deviance) | 0.0327 | N/A | 0.033 | 64.0 |
| Elastic-Net Regressor (L2 / Gamma Deviance) | 0.0327 | N/A | 0.033 | 64.0 |
| Keras Residual Neural Factorization Machine Regressor using Training Schedule (2 Layers: 96, 96 Units) | 0.0709 | N/A | 0.0711 | 64.0 |
| Keras Residual Cross Network Regressor using Training Schedule (3 Cross Layers, 4 Layers: 96, 96, 72, 72 Units) | 0.4313 | N/A | 0.4358 | 64.0 |

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., Gamma Deviance) 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 are two tables. The first contains a list of the final set of model feature variables, as well as summary statistics for the Keras Slim Residual Neural Network Regressor using Adaptive Training Schedule (1 Layer: 64 Units) model. The second table contains a detailed analysis of missing values.

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 |
| 시군구 | Text | 338 | 0 | N/A | N/A | N/A | N/A | N/A | N/A |
| 번지 | Categorical | 6555 | 184 | N/A | N/A | N/A | N/A | N/A | Low |
| 아파트명 | Text | 6522 | 1712 | N/A | N/A | N/A | N/A | N/A | N/A |
| 전용면적(㎡) | Numeric | 14218 | 0 | 77.16 | 29.38 | 81.86 | 10.02 | 424.32 | Low |
| 계약년월 | Numeric | 198 | 0 | 201475.9 | 418.81 | 201507.0 | 200701.0 | 202306.0 | Low |
| 건축년도 | Numeric | 60 | 0 | 1998.76 | 9.33 | 2000.0 | 1961.0 | 2023.0 | Low |
| target | Numeric | 12875 | 0 | 57962.59 | 46347.64 | 44750.0 | 500.0 | 1450000.0 | 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

The following table provides a summary of missing values. It includes the name of the feature, its type, a summary of the missing value count (both number of rows and as a percentage), and information on the type of imputation applied to the feature.

5.6.3.2 Data Quality Handling Report

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Feature Name | Var Type | Missing Count | Missing Percentage | Imputation Name | Imputation Description |
| 번지 | Categorical | 147 | 0 | One-Hot Encoding | Missing indicator treated as feature |
| 전용면적(㎡) | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 81.89 |
| 계약년월 | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 201507 |
| 건축년도 | Numeric | 0 | 0 | Missing Values Imputed | Imputed value: 2000 |

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 19.99996% 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 80.00004% of the data available for training, which is then divided into 5-folds for cross-validation and and red denotes the holdout sample.



6.1.1 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.

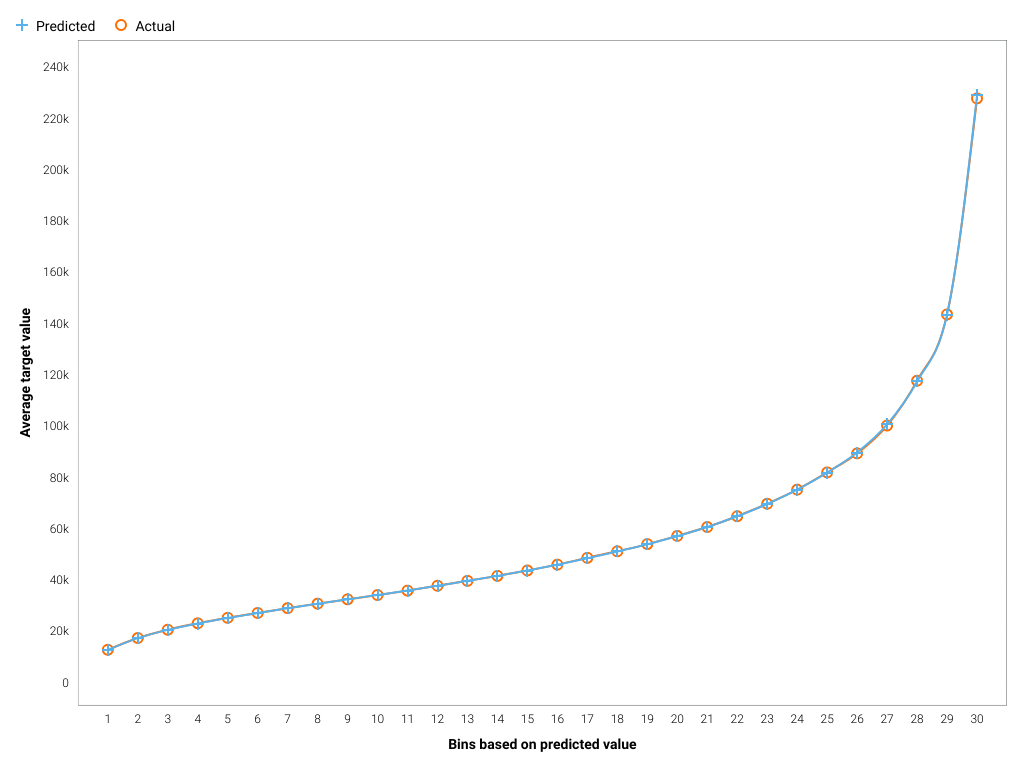
The performance metric used for this project was Gamma Deviance. The model performance results are presented below for out-of-sample testing:

|  |  |
| --- | --- |
| Scoring Type | Score (Gamma Deviance) |
| holdout | 0.0057 |
| validation | 0.0057 |

6.3 Sensitivity Testing and Analysis

6.3.1 Lift Chart

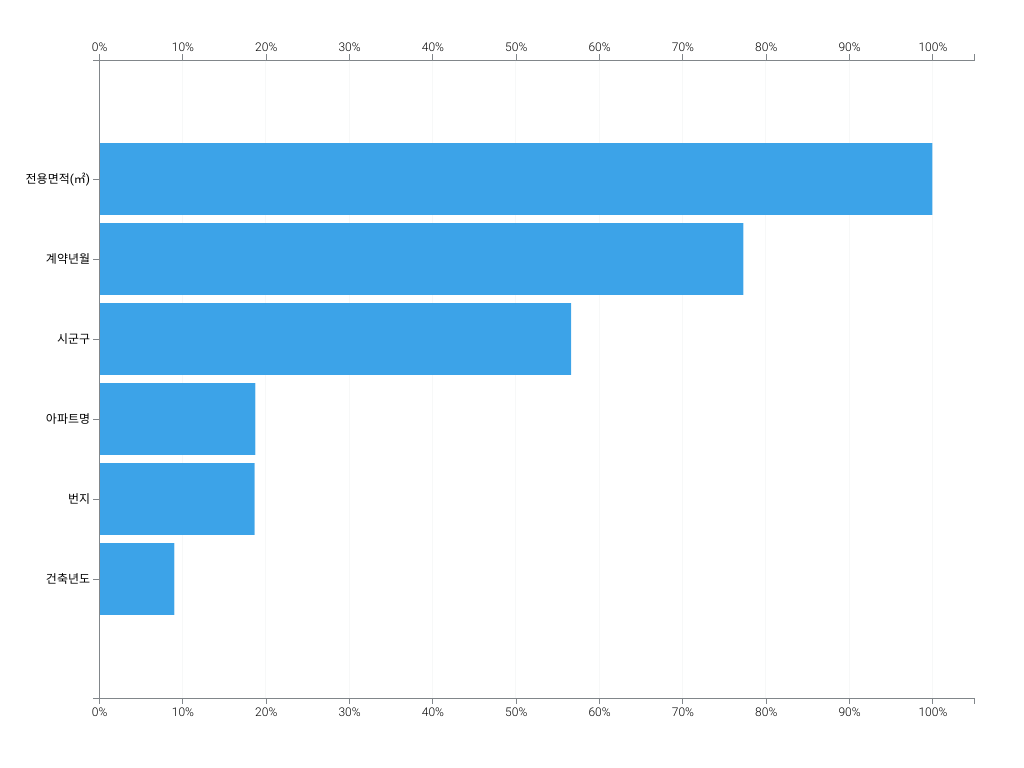
The Lift Chart sorts and groups numeric feature values into equal sized bins, depicting how well a model segments the target population and how capable it is of predicting the target. This helps the user to visualize model accuracy for each bin. The chart is sorted by predicted values -- lowest to highest predictions, for example -- which provides transparency to the model performance for different ranges of values of the target variable. Looking at the Lift Chart, the left side of the curve indicates where the model predicted a low score on one section of the population while the right side of the curve indicates where the model predicted a high score. The model Lift Chart is presented in the figure below.



The points on the Lift Chart indicate the average percentage in each bin. The "Predicted" blue line displays the average prediction score for the rows in that bin. The "Actual" orange line displays the actual percentage for the rows in that bin. In general, the steeper the Actual line is, and the more closely the Predicted line matches the actual line, the better the model. A close relationship between these two lines is indicative of the predictive accuracy of the model; a consistently increasing line is another good indicator of satisfactory model performance.

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.

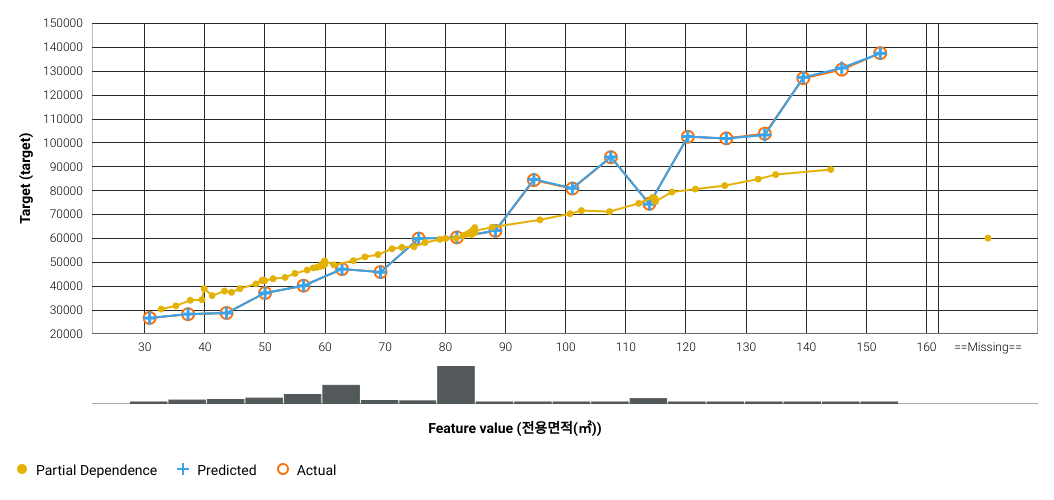


|  |  |  |
| --- | --- | --- |
| Feature Name | Impact Normalized | Impact Unnormalized |
| 전용면적(㎡) | 1.0 | 0.223 |
| 계약년월 | 0.7732 | 0.1724 |
| 시군구 | 0.5665 | 0.1263 |
| 아파트명 | 0.1875 | 0.0418 |
| 번지 | 0.1867 | 0.0416 |
| 건축년도 | 0.0903 | 0.0201 |

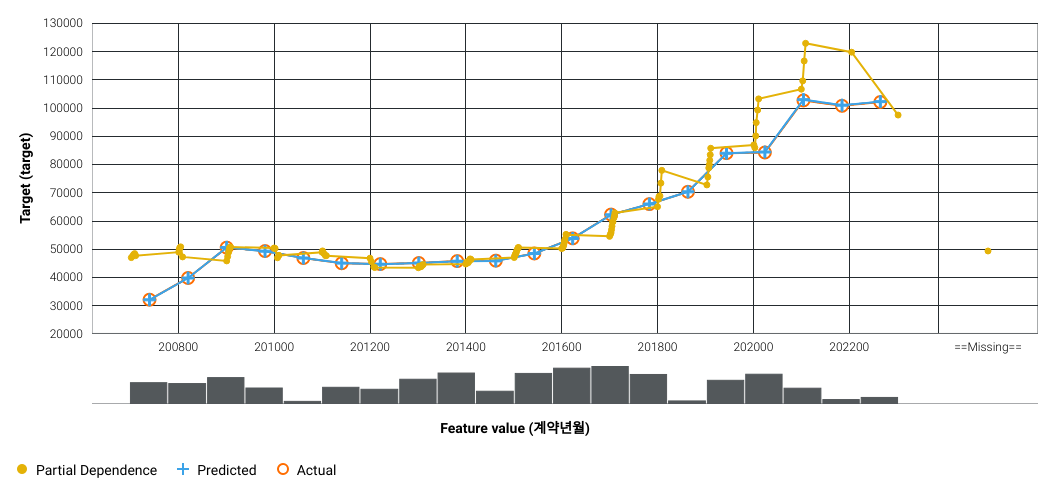
6.3.3 Sensitivity Analysis (Partial Dependence)

In the case of linear regression, users can gain considerable insight into the structure and interpretation of the model by examining its coefficients. For more complex models like support vector machines, random forests, or the blenders considered here, no comparably simple parametric description is available, making the interpretation of these models more difficult. To address this difficulty for his gradient boosting machine, Friedman (2001) proposed the use of partial dependence plots. Partial dependence plots show the average partial relationship between a set of predictors and the predicted response. The partial dependence plots below capture the top features in our model, as measured by Feature Impact.

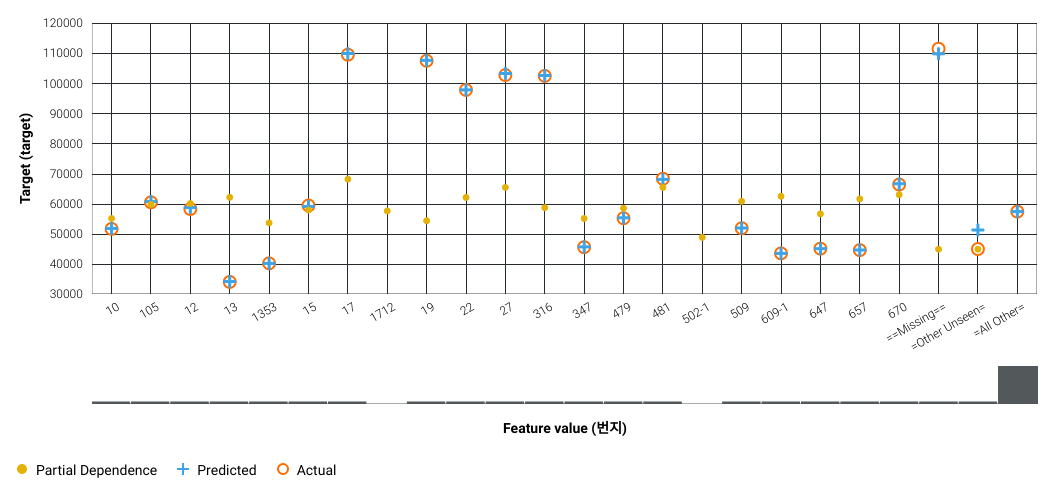
전용면적(㎡)



계약년월



번지



The orange circles depict, for the selected feature, the average target value for the aggregated feature values. The blue crosses depict, for the selected feature, the average prediction for a specific value. From the graph you can see that DataRobot also averages the predicted feature values. Comparing the actual and predicted points can identify segments where model predictions differ from observed data. This typically occurs when the segment size is small. In those cases, for example, some models may predict closer to the overall average.

The yellow partial dependence data points depict the marginal effect of a feature on the target variable after accounting for the average effects of all other predictive features. It indicates how, holding all other variables constant, the value of this feature affects prediction. DataRobot holds constant the values of all columns in the sample except the feature of interest. The value of the feature of interest is then reassigned to each possible value, calculating the average predictions for the sample at each setting. These values help determine how the value of each feature affects the target. The shape of the yellow data points describes the model's view of the marginal relationship between the selected feature and the target.

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.