# Driverless AI Experiment: 5.hatespeech\_vec\_janome

Generated by: yuki.shimada@h2o.ai

Generated on: Thu May 2 23:12:40 2024

[Driverless AI Experiment: 5.hatespeech\_vec\_janome 1](#_Toc55907577)

[Experiment Overview 1](#_Toc55907578)

[Data Overview 1](#_Toc55907579)

[Methodology 1](#_Toc55907580)

[Data Sampling 1](#_Toc55907581)

[Validation Strategy 2](#_Toc55907582)

[Model Tuning 2](#_Toc55907583)

[Feature Evolution 2](#_Toc55907584)

[Feature Transformations 2](#_Toc55907585)

[Final Model 2](#_Toc55907586)

[Alternative Models 2](#_Toc55907587)

[Deployment 2](#_Toc55907588)

[Partial Dependence Plots 2](#_Toc55907589)

[Shapley Values 2](#_Toc55907590)

[Appendix 3](#_Toc55907593)

## Experiment Overview

Driverless AI built None to predict label given 6 original features from the input dataset hatespeech\_train\_DAI.csv. This classification experiment completed in 11 minutes and 12 seconds (0:11:12), using 0 of the 6 original features, and 102 of the 102 engineered features.

### Performance

|  |  |
| --- | --- |
| **Dataset** | **F1** |
| Internal Validation | 0.562 |
| Test Data | 0.5 |

### System Specifications

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Address** | **System** | **System Memory** | **CPUs** | **GPUs** |
| http://127.0.0.1:12345 | Linux | 248 GB | 32 | 0 |

### Versions

|  |  |
| --- | --- |
|  |  |
| Driverless AI version | 1.10.7 |
| h2o4gpu version | 0.4.2 |
| h2o\_mli version | 1.10.26 |
| mojo2\_runtime version | 2.8.5 |
| daimojo version | 2.8.5 |
| daimojo version\_ppc64le | 2.8.1 |
| procsy version | 0.9.0 |
| pydatatable version | 1.1.0a2217 |
| vis\_data\_server version | 2.1.4 |

## Data Overview

This section provides information on the datasets used for the experiment.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **data** | **file path** | **file size** | **number of rows** | **number of columns** |
| training | /dai-data/yuki.shimada@h2o.ai/d008d1c0-084b-11ef-aabe-22b0f24f7d4a/hatespeech\_train\_DAI.csv.1714630717.8798265.bin | 2.3 MiB | 4,656 | 7 |
| validation | Not provided | None | None | None |
| testing | /dai-data/yuki.shimada@h2o.ai/c8cbd89e-084b-11ef-aabe-22b0f24f7d4a/hatespeech\_test\_DAI.csv.1714630705.7320695.bin | 307.0 KiB | 600 | 6 |

### Training Data

The training data consists of both numeric and categorical columns.

The summary of the columns is shown below:

#### Numeric Columns

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **name** | **logical\_type** | **storage\_type** | **min** | **mean** | **max** | **std** | **unique** | **freq of mode** |
| hold | N/A | int | 0.000 | 2.008 | 4.000 | 1.406 | 5 | 967 |

#### Boolean Columns

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **name** | **logical\_type** | **storage\_type** | **min** | **mean** | **max** | **std** | **freq of max value** |
| label | N/A | bool | False | 0.0582 | True | 0.2342 | 271 |

#### Categorical Columns

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **name** | **logical\_type** | **storage\_type** | **unique** | **top** | **freq of top value** |
| id | N/A | str | 4,656 | 001bb4f05 | 1 |
| source | N/A | str | 3 | newsplus | 1,749 |
| text | N/A | str | 4,656 | !aiはぇ～キャベツが一番頑張ってるんか、1位じゃない癖にイキってるホモサピエンスｗｗｗｗｗ | 1 |
| text\_token\_janome | catlabel, text | str | 4,656 | ! ai は ぇ ～ キャベツ が 一番 頑張っ てる ん か 、 1 位 じゃ ない 癖 に イキ | 1 |
| text\_token\_tohokuBertBase | N/A | str | 4,656 | ! a ##i は ぇ ~ キャ ##ベツ が 一番 頑 ##張っ てる ん か 、 1 位 じゃ | 1 |

#### Shifts Detected

Driverless AI can perform shift detection between the training, validation, and testing datasets. It does this by training a binomial model to predict which dataset a record belongs to. For example, it may find that it is able to separate the training and testing data with an AUC of 0.8 using only the column: C1 as the predictor. This indicates that there is some sort of drift in the distribution of C1 between the training and testing data.

For this experiment, Driverless AI checked the train and test data for any shift in distribution and found the following significant differences:

* Significant difference detected between training and test data distribution for feature <<<text>>> (AUC: 1.0). Dropping this feature.
* Significant difference detected between training and test data distribution for feature <<<id>>> (AUC: 1.0). Dropping this feature.
* Significant difference detected between training and test data distribution for feature <<<text\_token\_tohokuBertBase>>> (AUC: 1.0). Dropping this feature.

## Methodology

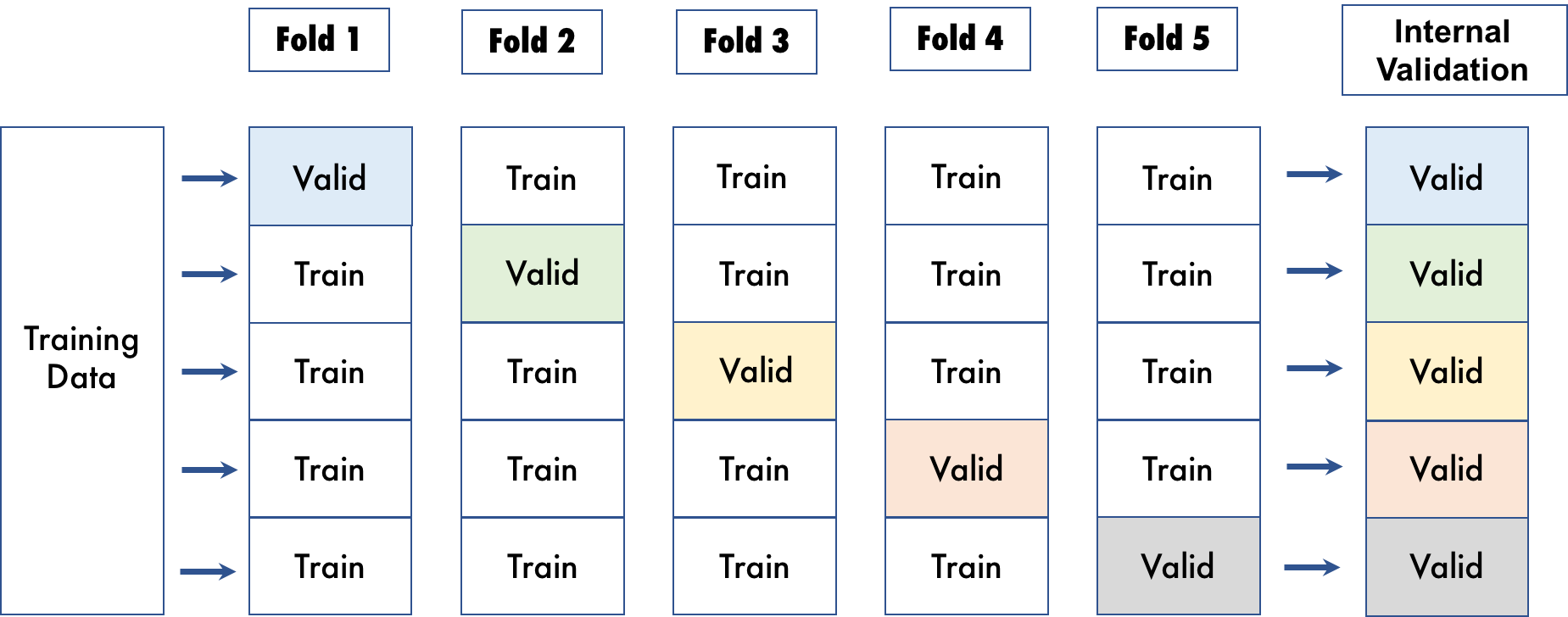
This section describes the experiment methodology.

## Data Sampling

In Driverless AI, data sampling is a pre-processing step that is done before model training begins; it is not related to sampling done during model training. Driverless AI does not perform data sampling unless the dataset is big or highly imbalanced. Whether a dataset is considered big depends on the experiment’s accuracy setting and the statistical\_threshold\_data\_size\_large config.toml parameter.

Driverless AI did not perform any down sampling of the data.

## Validation Strategy

Driverless AI automatically split the training data to determine the performance of the model parameter tuning and feature engineering stages. For the experiment, Driverless AI split the data into 3-fold cross validation, using stratified sampling. With cross validation, the whole dataset is utilized by training 3 models where each model is trained on a different subset of the training data. The visualization below shows how cross validation is utilized to get predictions on hold out data. The visualization shows an example of cross validation with 5 folds. For this experiment, however, 3 folds were created. Note: The cross-validation process was repeated 4 times to ensure the validation metrics are robust since the training data was small. 

## Model Tuning

The table below shows the score and training time of the xgboost, constant and lightgbm models evaluated by Driverless AI. The table shows the parameter tuning models evaluated, ordered based on a combination of largest score and lowest training time.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **job order** | **booster** | **nfeatures** | **training times** | **scores (F1)** |
| 5 | gbtree | 152 | 14.4353 | 0.5304 |
| 6 | lightgbm | 152 | 8.4487 | 0.5129 |
| 4 | lightgbm | 50 | 2.2519 | 0.4078 |
| 1 | gbtree | 50 | 7.2555 | 0.4038 |
| 0 | lightgbm | 50 | 4.2667 | 0.3899 |
| 3 | gbtree | 50 | 2.8176 | 0.11 |
| 7 | constant | 1 | 0.6233 | 0.11 |

More detailed information on the parameters evaluated for each algorithm is shown below.

### gbtree tuning

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **tree method** | **grow policy** | **max depth** | **max leaves** | **colsample bytree** | **subsample** | **nfeatures** | **training times** | **scores (F1)** |
| hist | depthwise | 8.0 | 256.0 | 0.8 | 0.7 | 152 | 14.4353 | 0.5304 |
| hist | depthwise | 8.0 | 256.0 | 0.8 | 0.7 | 50 | 7.2555 | 0.4038 |
| hist | lossguide | 9.0 | 512.0 | 0.6 | 0.7 | 50 | 2.8176 | 0.11 |

### constant tuning

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **job order** | **booster** | **nfeatures** | **training times** | **scores (F1)** |
| 7 | constant | 1 | 0.6233 | 0.11 |

### lightgbm tuning

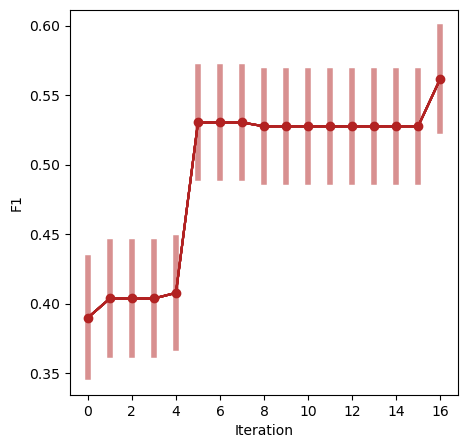
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **tree method** | **grow policy** | **max depth** | **max leaves** | **colsample bytree** | **subsample** | **nfeatures** | **training times** | **scores (F1)** |
|  | depthwise | 8.0 | 256.0 | 0.8 | 0.7 | 152 | 8.4487 | 0.5129 |
|  | lossguide | 40.0 | 32.0 | 0.7 | 0.9 | 50 | 2.2519 | 0.4078 |
|  | depthwise | 8.0 | 256.0 | 0.8 | 0.7 | 50 | 4.2667 | 0.3899 |

## Feature Evolution

During the Model and Feature Tuning Stage, Driverless AI evaluates the effects of different types of algorithms, algorithm parameters, and features. The goal of the Model and Feature Tuning Stage is to determine the best algorithm and parameters to use during the Feature Evolution Stage.

In the Feature Evolution Stage, Driverless AI trained xgboost, constant and lightgbm models (204) where each model evaluated a different set of features. The Feature Evolution Stage uses a genetic algorithm to search the large feature engineering space.

The graph below shows the effect the Model and Feature Tuning Stage and Feature Evolution Stage had on the performance.



Based on the experiment settings and column types in the dataset, Driverless AI was able to explore the following transformers:

* **BinnerTransformer**: None
* **ClusterDistTransformer**: the Cluster Distance Transformer clusters selected numeric columns and uses the distance to a specific cluster as a new feature.
* **ClusterTETransformer**: the Cluster Target Encoding Transformer clusters selected numeric columns and calculates the mean of the response column for each cluster. The mean of the response is used as a new feature. Cross Validation is used to calculate mean response to prevent overfitting.
* **InteractionsTransformer**: the Interactions Transformer adds, divides, multiplies, and subtracts two numeric columns in the data to create a new feature.
* **NumToCatTETransformer**: the Numeric to Categorical Target Encoding Transformer converts numeric columns to categoricals by binning and then calculates the mean of the response column for each group. The mean of the response for the bin is used as a new feature. Cross Validation is used to calculate mean response to prevent overfitting.
* **NumToCatWoETransformer**: the Numeric to Categorical Weight of Evidence Transformer converts a numeric column to categorical by binning and then calculates Weight of Evidence for each bin. The Weight of Evidence is used as a new feature. Weight of Evidence measures the “strength” of a grouping for separating good and bad risk and is calculated by taking the log of the ratio of distributions for a binary response column.
* **TruncSVDNumTransformer**: the Truncated SVD Transformer trains a Truncated SVD model on selected numeric columns and uses the components of the truncated SVD matrix as new features.
* **CatOriginalTransformer**: the Categorical Original Transformer applies an identity transformation that leaves categorical features as they are. This transformer works with models that can handle non-numeric feature values.
* **CVTargetEncodeTransformer**: the Cross Validation Target Encoding Transformer calculates the mean of the response column for each value in a categorical column and uses this as a new feature. Cross Validation is used to calculate mean response to prevent overfitting.
* **FrequentTransformer**: the Frequent Transformer calculates the frequency for each value in categorical column(s) and uses this as a new feature. This count can be either the raw count or the normalized count.
* **WeightOfEvidenceTransformer**: the Weight of Evidence Transformer calculates Weight of Evidence for each value in categorical column(s). The Weight of Evidence is used as a new feature. Weight of Evidence measures the “strength” of a grouping for separating good and bad risk and is calculated by taking the log of the ratio of distributions for a binary response column.
* **CatTransformer**: the Categorical Transformer sorts a categorical column in lexicographical order and uses the order index created as a new feature. This transformer works with models that can handle categorical features.
* **OneHotEncodingTransformer**: the One-hot Encoding transformer converts a categorical column to a series of boolean features by performing one-hot encoding. The boolean features are used as new features.
* **OneHotEncodingUnsupervisedTransformer**: None
* **IsHolidayTransformer**: the Is Holiday Transformer determines if a date column is a holiday. A boolean column indicating if the date is a holiday is added as a new feature. Creates a separate feature for holidays in the United States, United Kingdom, Germany, Mexico, and the European Central Bank. Other countries available in the python Holiday package can be added via the configuration file.
* **DateTimeDiffTransformer**: None
* **DatesTransformer**: the Date Transformer retrieves any date or time values, including: Year, Quarter, Month, Day, Day of Year, Week, Weekday, Hour, Minute, Second.
* **TextTransformer**: the Text Transformer tokenizes a text column and creates a TFIDF matrix (term frequency-inverse document frequency) or count (count of the word) matrix. This may be followed by dimensionality reduction using truncated SVD. Selected components of the TF-IDF/Count matrix are used as new features.
* **TextLinModelTransformer**: the Text Linear Model Transformer trains a linear model on a TF-IDF matrix created from a text feature to predict the response column. The linear model prediction is used as a new feature. Cross Validation is used when training the linear model to prevent overfitting.
* **TextOriginalTransformer**: None
* **TimeSeriesTargetEncTransformer**: None
* **ImageVectorizerTransformer**: the Image Vectorizer Transformer converts a column with an image path or URI to the embeddings representation. It utilizes ImageNet pre-trained models with a possible finetuning on the input data.
* **ImageOriginalTransformer**: None
* **CVCatNumEncodeTransformer**: the Cross Validation Categorical to Numeric Encoding Transformer calculates an aggregation of a numeric column for each value in a categorical column (ex: calculate the mean Temperature for each City) and uses this aggregation as a new feature.
* **NumCatTETransformer**: the Numeric Categorical Target Encoding Transformer calculates the mean of the response column for several selected columns. If one of the selected columns is numeric, it is first converted to categorical by binning. The mean of the response column is used as a new feature. Cross Validation is used to calculate mean response to prevent overfitting.

**Dropped Features**

Below is the complete list of dropped features due to numerous reasons.

|  |  |
| --- | --- |
| **Name** | **Reason** |
| hold | User Dropped |
| id | DAI Data Shift |
| source | User Dropped |
| text | DAI Data Shift |
| text\_token\_tohokuBertBase | DAI Data Shift |

## Feature Transformations

The result of the Feature Evolution Stage is a set of features to use for the final model. Some of these features were automatically created by Driverless AI. The top features used in the final model are shown below, ordered by importance. The features in the table are limited to the top 50, restricted to those with relative importance greater than or equal to 0.003. If no transformer was applied, the feature is an original column.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Feature** | **Description** | **Transformer** | **Relative Importance** |
| 1 | 3\_TxtTE: text\_token\_janome.0 | Predicted probabilities of class #1 based on linear model on Tfidf features from text column ['text\_token\_janome'] [internal parameters:(5000, 1, 0.01)] | Bag of Words Regression | 1.0 |
| 2 | 1\_TxtTE: text\_token\_janome.0 | Predicted probabilities of class #1 based on linear model on Tfidf features from text column ['text\_token\_janome'] [internal parameters:(5000, 1, 10)] | Bag of Words Regression | 0.3439 |
| 3 | 0\_Txt: text\_token\_janome.0 | Feature #1 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0344 |
| 4 | 2\_Txt: text\_token\_janome.16 | Feature #17 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.0233 |
| 5 | 2\_Txt: text\_token\_janome.10 | Feature #11 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.0232 |
| 6 | 0\_Txt: text\_token\_janome.15 | Feature #16 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0209 |
| 7 | 0\_Txt: text\_token\_janome.24 | Feature #25 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0193 |
| 8 | 0\_Txt: text\_token\_janome.47 | Feature #48 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0185 |
| 9 | 0\_Txt: text\_token\_janome.33 | Feature #34 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0182 |
| 10 | 0\_Txt: text\_token\_janome.43 | Feature #44 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0179 |
| 11 | 0\_Txt: text\_token\_janome.5 | Feature #6 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0178 |
| 12 | 0\_Txt: text\_token\_janome.37 | Feature #38 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0175 |
| 13 | 0\_Txt: text\_token\_janome.38 | Feature #39 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.017 |
| 14 | 2\_Txt: text\_token\_janome.49 | Feature #50 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.0168 |
| 15 | 0\_Txt: text\_token\_janome.20 | Feature #21 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0158 |
| 16 | 0\_Txt: text\_token\_janome.35 | Feature #36 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0155 |
| 17 | 2\_Txt: text\_token\_janome.45 | Feature #46 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.0153 |
| 18 | 0\_Txt: text\_token\_janome.34 | Feature #35 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0153 |
| 19 | 0\_Txt: text\_token\_janome.1 | Feature #2 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0152 |
| 20 | 2\_Txt: text\_token\_janome.0 | Feature #1 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.015 |
| 21 | 0\_Txt: text\_token\_janome.27 | Feature #28 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0149 |
| 22 | 0\_Txt: text\_token\_janome.29 | Feature #30 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0148 |
| 23 | 0\_Txt: text\_token\_janome.19 | Feature #20 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0148 |
| 24 | 0\_Txt: text\_token\_janome.11 | Feature #12 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0148 |
| 25 | 0\_Txt: text\_token\_janome.2 | Feature #3 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0144 |
| 26 | 0\_Txt: text\_token\_janome.14 | Feature #15 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0138 |
| 27 | 0\_Txt: text\_token\_janome.21 | Feature #22 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0134 |
| 28 | 2\_Txt: text\_token\_janome.35 | Feature #36 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.0132 |
| 29 | 0\_Txt: text\_token\_janome.22 | Feature #23 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0132 |
| 30 | 2\_Txt: text\_token\_janome.3 | Feature #4 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.0131 |
| 31 | 0\_Txt: text\_token\_janome.36 | Feature #37 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0128 |
| 32 | 2\_Txt: text\_token\_janome.13 | Feature #14 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.0128 |
| 33 | 2\_Txt: text\_token\_janome.25 | Feature #26 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.0125 |
| 34 | 2\_Txt: text\_token\_janome.11 | Feature #12 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.0125 |
| 35 | 0\_Txt: text\_token\_janome.12 | Feature #13 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0123 |
| 36 | 0\_Txt: text\_token\_janome.7 | Feature #8 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.012 |
| 37 | 0\_Txt: text\_token\_janome.31 | Feature #32 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0118 |
| 38 | 2\_Txt: text\_token\_janome.22 | Feature #23 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.0118 |
| 39 | 2\_Txt: text\_token\_janome.19 | Feature #20 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.0107 |
| 40 | 2\_Txt: text\_token\_janome.14 | Feature #15 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.0106 |
| 41 | 2\_Txt: text\_token\_janome.1 | Feature #2 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.0105 |
| 42 | 0\_Txt: text\_token\_janome.17 | Feature #18 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0103 |
| 43 | 0\_Txt: text\_token\_janome.16 | Feature #17 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0101 |
| 44 | 2\_Txt: text\_token\_janome.12 | Feature #13 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.01 |
| 45 | 2\_Txt: text\_token\_janome.32 | Feature #33 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.01 |
| 46 | 0\_Txt: text\_token\_janome.46 | Feature #47 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0094 |
| 47 | 2\_Txt: text\_token\_janome.4 | Feature #5 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.0092 |
| 48 | 0\_Txt: text\_token\_janome.10 | Feature #11 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0091 |
| 49 | 2\_Txt: text\_token\_janome.40 | Feature #41 of count-based word embedding into 5000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(5000, False, 2, 50)] | Bag of Words | 0.009 |
| 50 | 0\_Txt: text\_token\_janome.32 | Feature #33 of tf-idf-based word embedding into 1000 components (followed by dimensionality reduction to 50 dimensions) of 'text\_token\_janome' [internal parameters:(1000, True, 1, 50)] | Bag of Words | 0.0089 |



### Advanced NLP and Image Transformations

Driverless AI used advanced NLP/Image transformations in the final model pipeline. Detailed description of each advanced transformer includes the assumptions, limitations and the architecture of the transformer. There may be multiple transformers of the same type in the final pipeline having different architectures.

#### TextTransformer

Term Frequency - Inverse Document Frequency (TF-IDF) Vectorizer creates a representation of texts by constructing a one-hot matrix of word n-grams using the TF-IDF values and then reducing the dimensionality using Truncated SVD.

**Assumptions**

The order and the position of the words in documents are not relevant for the classification/regression.

**Limitations**

The vocabulary only includes n-grams present in the training set. The algorithm ignores word order. The algorithm cannot take advantage of pretrained word embeddings.

**Sampling**

Only the top 1000 most frequent tokens are included in the vocabulary. Tokens with absolute frequency < 3 or relative frequency > 0.9 across the documents are not included in the vocabulary.

**Architecture**

1. Term Frequency - Inverse Document Frequency (TF-IDF) Vectorizer parameters:  
 token\_pattern='\b\w+\b',  
 ngram\_range=(1,1),  
 max\_features=1000,  
 min\_df=3,  
 max\_df=0.9,  
 lowercase=True,  
 stop\_words=None  
  
2. Truncated SVD parameters:  
 n\_dimension=50

#### TextLinModelTransformer

Term Frequency - Inverse Document Frequency (TF-IDF) vectorizer creates a representation of texts by constructing a one-hot matrix of word n-grams using the TF-IDF values. These features are used as inputs to get cross-validated predictions from logistic regression model.

**Assumptions**

The order and the position of the words in documents are not relevant for the classification/regression.

**Limitations**

The vocabulary only includes n-grams present in the training set. The algorithm ignores word order. The algorithm cannot take advantage of pretrained word embeddings.

**Sampling**

Tokens with absolute frequency < 3 or relative frequency > 0.9 across the documents are not included in the vocabulary.

**Architecture**

1. TF-IDF parameters:  
 analyzer=word  
 binary=False  
 decode\_error=strict  
 dtype=<class 'numpy.float64'>  
 encoding=utf-8  
 input=content  
 lowercase=True  
 max\_df=0.9  
 max\_features=5000  
 min\_df=3  
 ngram\_range=(1, 1)  
 norm=l2  
 preprocessor=None  
 smooth\_idf=1  
 stop\_words=None  
 strip\_accents=unicode  
 sublinear\_tf=1  
 token\_pattern=(\b\w+\b|[!"#$%&'()\*+,-./:;<=>?@\[\]^\_`{|}~“”¨«»®´·º½¾¿¡§£₤‘’])  
 tokenizer=None  
 use\_idf=1  
 vocabulary=None  
  
2. Logistic regression parameters:  
 C=10,  
 dual=False

#### TextTransformer

Count Vectorizer creates a representation of texts by constructing a one-hot matrix of word n-grams using the frequency values and then reducing the dimensionality using Truncated SVD.

**Assumptions**

The order and the position of the words in documents are not relevant for the classification/regression.

**Limitations**

The vocabulary only includes n-grams present in the training set. The algorithm ignores word order. The algorithm cannot take advantage of pretrained word embeddings.

**Sampling**

Only the top 5000 most frequent tokens are included in the vocabulary. Tokens with absolute frequency < 3 or relative frequency > 0.9 across the documents are not included in the vocabulary.

**Architecture**

1. Count Vectorizer parameters:  
 token\_pattern='\b\w+\b',  
 ngram\_range=(1,2),  
 max\_features=5000,  
 min\_df=3,  
 max\_df=0.9,  
 lowercase=True,  
 stop\_words=None  
  
2. Truncated SVD parameters:  
 n\_dimension=50

#### TextLinModelTransformer

Term Frequency - Inverse Document Frequency (TF-IDF) vectorizer creates a representation of texts by constructing a one-hot matrix of word n-grams using the TF-IDF values. These features are used as inputs to get cross-validated predictions from logistic regression model.

**Assumptions**

The order and the position of the words in documents are not relevant for the classification/regression.

**Limitations**

The vocabulary only includes n-grams present in the training set. The algorithm ignores word order. The algorithm cannot take advantage of pretrained word embeddings.

**Sampling**

Tokens with absolute frequency < 3 or relative frequency > 0.9 across the documents are not included in the vocabulary.

**Architecture**

1. TF-IDF parameters:  
 analyzer=word  
 binary=False  
 decode\_error=strict  
 dtype=<class 'numpy.float64'>  
 encoding=utf-8  
 input=content  
 lowercase=True  
 max\_df=0.9  
 max\_features=5000  
 min\_df=3  
 ngram\_range=(1, 1)  
 norm=l2  
 preprocessor=None  
 smooth\_idf=1  
 stop\_words=None  
 strip\_accents=unicode  
 sublinear\_tf=1  
 token\_pattern=(\b\w+\b|[!"#$%&'()\*+,-./:;<=>?@\[\]^\_`{|}~“”¨«»®´·º½¾¿¡§£₤‘’])  
 tokenizer=None  
 use\_idf=1  
 vocabulary=None  
  
2. Logistic regression parameters:  
 C=0.01,  
 dual=False

**Monotonicity Constraints Details**

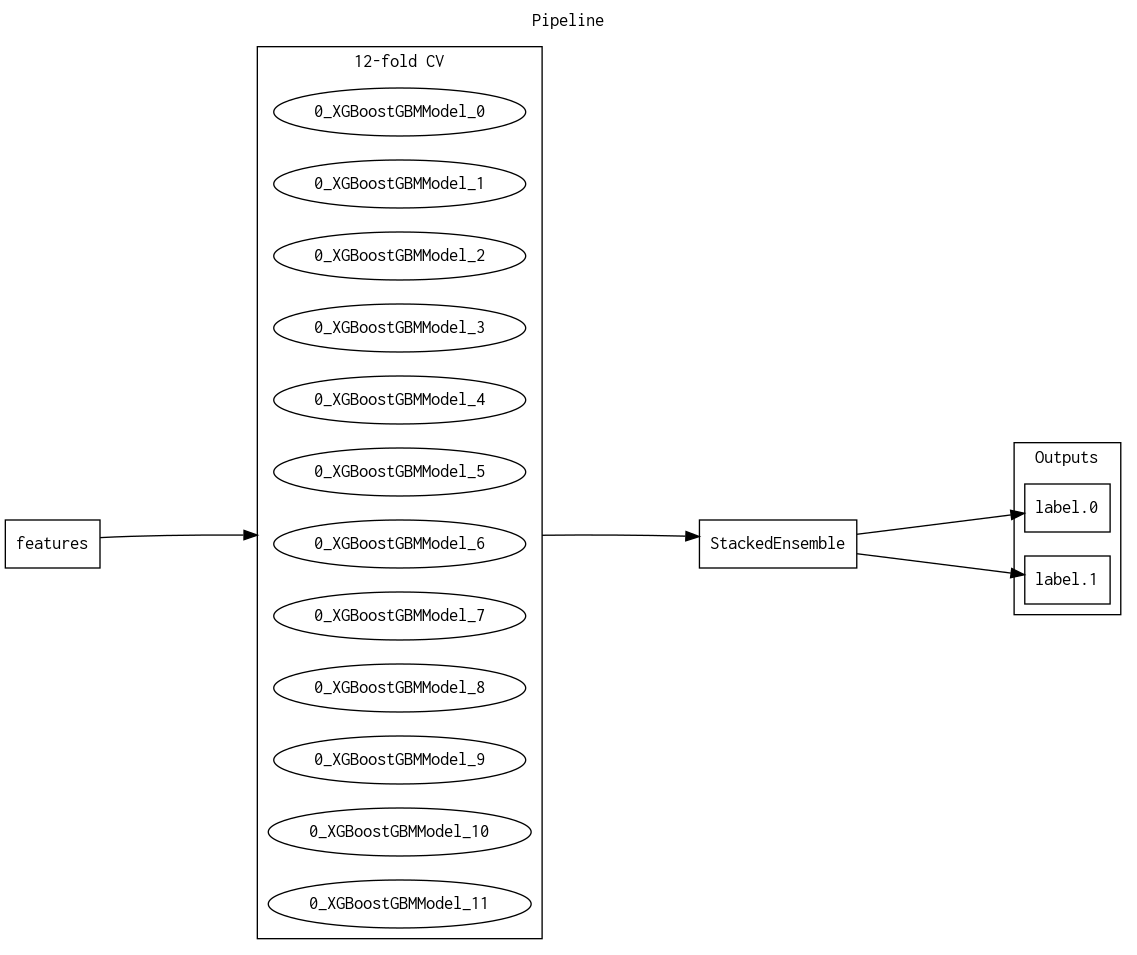
Monotonicity constraints enforce a monotonic relationship between a specified feature and the target prediction. For example, given a model trained to predict housing prices, we might want to enforce that the model predicts higher housing prices with increasing lot size and lower housing prices with increasing neighborhood crime rate.

This experiment enables automatic monotonicity constraints, which mean Driverless AI automatically determines if monotonicity is present and then enforces it through all or part of the modeling pipelines. Depending on the level of feature-target correlation, Driverless AI assigns positive, negative, or no monotonicity constraints. Specifically, monotonicity constraints are enforced if the absolute correlation is greater than 0.1 (the monotonicity correlation threshold set for this experiment). Driverless AI does not enforce monotonicity constraints for features below this correlation threshold.

## Final Model

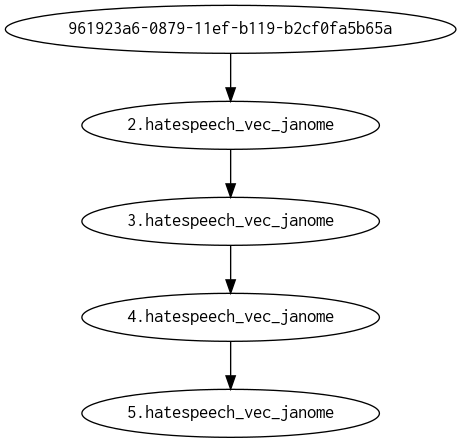
**Pipeline**

Final StackedEnsemble pipeline with ensemble\_level=4 transforming 1 original features -> 102 features in each of 5 models each fit on 12 internal holdout splits then linearly blended:



**Model Lineage**

The following plot shows the experiment lineage for the current experiment 5.hatespeech\_vec\_janome.



**Details**

* The fitted features of the final model are the best features found during the feature engineering iterations.
* The target transformer indicates the type of transformation applied to the target column.

For a complete list of the parameters of the final model, see the Appendix.

**Performance of Final Model**

Some metrics below require the probabilities to be converted into labels. Predicted labels are generated based on some probability threshold. For example, we may say the label is “YES” if the probability is greater than 0.5.

Choosing this threshold can greatly change the performance metric. This can be seen in the ROC and Precision Recall curves on the Driverless AI UI, where different thresholds on the curve are highlighted, that marks the best Accuracy, the best MCC, and the best F1.

In the Performance Table below, some metrics that require labels have been calculated so that the threshold chosen optimizes the metric. The metrics whose thresholds are optimized are: Accuracy, MCC, F1, F05, and F2. Other metrics that require labels like Precision and Recall choose the threshold that optimizes the F1 because these are trivial to optimize (i.e, Recall will always be perfect when the threshold is 0).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Scorer** | **Optimized** | **Better score is** | **Final ensemble scores on validation (internal or external holdout(s)) data** | **Final ensemble standard deviation on validation (internal or external holdout(s)) data** | **Final test scores** | **Final test standard deviation** |
| F1 | \* | higher | 0.5616479 | 0.03724536 | 0.5 | 0.06450092 |
| ACCURACY |  | higher | 0.9488134 | 0.004984356 | 0.9516667 | 0.008214148 |
| AUC |  | higher | 0.9305343 | 0.01187822 | 0.9383565 | 0.01264288 |
| AUCPR |  | higher | 0.4730106 | 0.05048291 | 0.4968062 | 0.08377116 |
| F05 |  | higher | 0.5374472 | 0.04376805 | 0.5652174 | 0.0888248 |
| F2 |  | higher | 0.664242 | 0.03353025 | 0.6220096 | 0.04949617 |
| FDR |  | lower | 0.520767 | 0.0529614 | 0.6231884 | 0.167258 |
| FNR |  | lower | 0.3088914 | 0.06996139 | 0.2571429 | 0.1541095 |
| FOR |  | lower | 0.01951956 | 0.004485849 | 0.01694915 | 0.009116889 |
| FPR |  | lower | 0.04745773 | 0.01187952 | 0.07610619 | 0.03003687 |
| GINI |  | higher | 0.8610686 | 0.02375643 | 0.876713 | 0.02528576 |
| LOGLOSS |  | lower | 0.1346325 | 0.01183782 | 0.1367877 | 0.01706953 |
| MACROAUC |  | higher | 0.9305343 | 0.01187822 | 0.9383565 | 0.01264288 |
| MACROF1 |  | higher | 0.5616479 | 0.03724536 | 0.5 | 0.06450092 |
| MACROMCC |  | higher | 0.5443247 | 0.03613864 | 0.4898353 | 0.06351579 |
| MCC |  | higher | 0.5443247 | 0.03613864 | 0.4898353 | 0.06351579 |
| NPV |  | higher | 0.9804804 | 0.004485849 | 0.9830508 | 0.009116889 |
| PRECISION |  | higher | 0.479233 | 0.0529614 | 0.3768116 | 0.167258 |
| RECALL |  | higher | 0.6911086 | 0.06996139 | 0.7428571 | 0.1541095 |
| TNR |  | higher | 0.9525423 | 0.01187952 | 0.9238938 | 0.03003687 |

## Alternative Models

During the experiment, Driverless AI trained 25 alternative models. The following algorithms were evaluated during the Driverless AI experiment:

|  |  |  |  |
| --- | --- | --- | --- |
| **algorithm** | **package** | **version** | **documentation** |
| gbtree | xgboost | 1.5.0-dev | XGBoost: eXtreme Gradient Boosting library. Contributors: https://github.com/dmlc/xgboost/blob/master/CONTRIBUTORS.md |
| constant | custom package | 1.10.7 | reference model that predicts a constant aimed at minimizing the given scorer |
| lightgbm | lightgbm | 3.3.2.99 | LightGBM, Light Gradient Boosting Machine. Contributors: https://github.com/microsoft/LightGBM/graphs/contributors. |

Driverless AI can evaluate an array of algorithms, including but not limited to XGBoost GBM, XGBoost Dart, XGBoost GLM, LightGBM, RuleFit, Tensorflow, and FTRL models. The table below explains why certain algorithms were not selected for the final model, if any.

|  |  |
| --- | --- |
| **algorithm** | **selection** |
| gblinear | algorithm not evaluated due to experiment configuration |
| decision tree | algorithm not evaluated due to experiment configuration |
| rulefit | algorithm not evaluated due to experiment configuration |
| tensorflow | algorithm not evaluated due to experiment configuration |
| ftrl | algorithm not evaluated due to experiment configuration |
| dart | algorithm not evaluated due to experiment configuration |
| gbtree | not selected due to low performance during feature evolution stage |
| lightgbm | not selected due to low performance during feature evolution stage |

## Deployment

For this experiment, both Python and MOJO Scoring Pipelines are available for productionizing the final model pipeline for a given row of data or table of data.

### Python Scoring Pipeline

This package contains an exported model and Python 3.6 source code examples for productionizing models built using H2O Driverless AI. The Python Scoring Pipeline is located here:

* **yuki.shimada@h2o.ai/h2oai\_experiment\_21671708-08d7-11ef-810f-760ec22f4b3a/scoring\_pipeline/scorer.zip**

The files in this package allow you to transform and score on new data in a couple of different ways:

* From Python 3.6, you can import a scoring module, then use the module to transform and score on new data.
* From other languages and platforms, you can use the TCP/HTTP scoring service bundled with this package to call into the scoring pipeline module through remote procedure calls (RPC).

### MOJO Scoring Pipeline

Note: The MOJO Scoring Pipeline is currently in a beta state. Updates and improvements will continue to be made in subsequent Driverless AI releases. The MOJO Scoring Pipeline is located here:

* **yuki.shimada@h2o.ai/h2oai\_experiment\_21671708-08d7-11ef-810f-760ec22f4b3a/mojo\_pipeline/mojo.zip**

For completed experiments, Driverless AI converts models to MOJOs (Model Objects, Optimized). A MOJO is a scoring engine that can be deployed in any Java environment for scoring in real time.

## Partial Dependence Plots

Partial dependence plots show the partial dependence as a function of specific values for a feature subset. The plots show how machine-learned response functions change based on the values of an input feature of interest, while taking nonlinearity into consideration and averaging out the effects of all other input features. Partial dependence plots enable increased transparency in a model and enable the ability to validate and debug a model by comparing a feature’s average predictions across its domain to known standards and reasonable expectations.

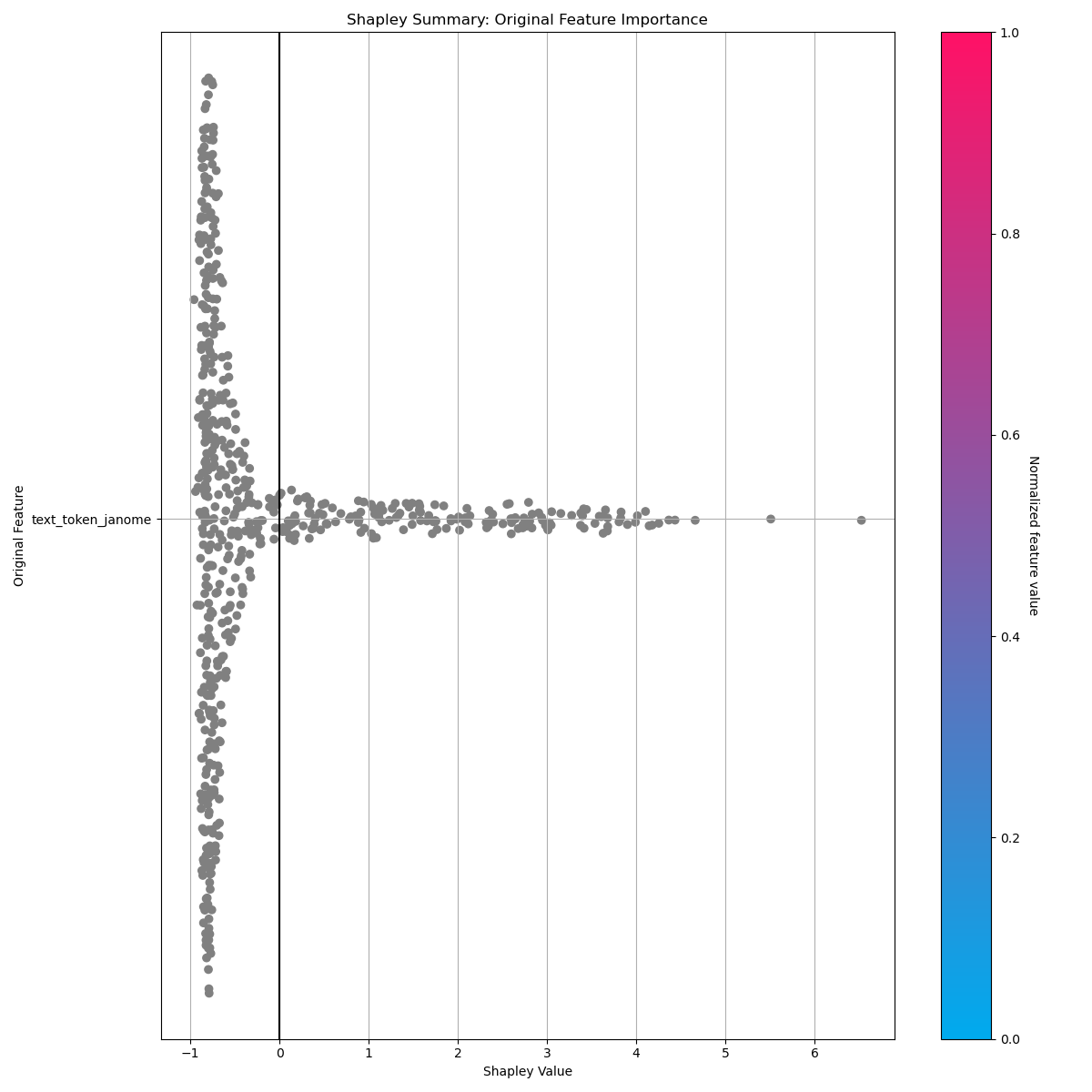
The partial dependence plots are shown for the top 0 original variables. The top 0 original variables are chosen based on their Component Based Variable Importance.

## Shapley Values

#### Shapley Contributions on the Test Dataset

Shapley explanations are a technique with credible theoretical support that presents consistent global and local feature contributions. For regression problems, local Shapley feature contributions plus the bias term sum to the final model’s prediction. For classification problems, they sum to the prediction before applying the link function.

This section uses Driverless AI’s Naive Shapley method to calculate local Shapley explanations for original features. These explanations are approximation for the original features and are based on how often the features are used in transformed features, and how important those transformed features are to the final model. The importance of each transformed feature is distributed equally to all original features that helped create it. This is then summed for each original feature.



## Appendix