CASE STUDY – 1

NISHA -2211566

What algorithms Azure Supports? Give a list of all Hyperparameters that all these algorithms support? Explain with examples on how the following algorithms work?

- One v/s One Multiclass
- One v/s All Multiclass

Algorithms supported by Azure

- ## Two-class classification

1. **Logistic Regression -**Logistic regression is a well-known statistical technique that is used for modeling many kinds of problems. This algorithm is a *supervised learning* method; therefore, you must provide a dataset that already contains the outcomes to train the model.

Module parameters

Name	Range	Type	Default	Description
Optimization tolerance	>=double.Epsilon	Float	0.0000001	Specify a tolerance value for the L-BFGS optimizer
L1 regularization weight	>=0.0	Float	1.0	Specify the L1 regularization weight
L2 regularization weight	>=0.0	Float	1.0	Specify the L2 regularization weight
Memory size for L-BFGS	>=1	Integer	20	Specify the amount of memory (in MB) to use for the L-BFGS optimizer
Random number seed	Any	Integer		Type a value to seed the random number generator used by the model. Leave it blank for the default.
Allow unknown categorical levels	Any	Boolean	True	Indicate whether an additional level should be created for each categorical column. Any levels in the test dataset that are not available in the training dataset are mapped to this additional level.

2. **Decision Forest**
   This decision forest algorithm is an ensemble learning method intended for classification tasks. Ensemble methods are based on the general principle that rather than relying on a single model, you can get better results and a more generalized model by creating multiple related models and combining them in some way. Generally, ensemble models provide better coverage and accuracy than single decision trees.

Module parameters

Name	Range	Type	Default	Description
Resampling method	Any	ResamplingMethod	Bagging	Choose a resampling method
Number of decision trees	>=1	Integer	8	Specify the number of decision trees to create in the ensemble
Maximum depth of the decision trees	>=1	Integer	32	Specify the maximum depth of any decision tree that can be created
Number of random splits per node	>=1	Integer	128	Specify the number of splits generated per node, from which the optimal split is selected
Minimum number of samples per leaf node	>=1	Integer	1	Specify the minimum number of training samples that are required to produce a leaf node
Allow unknown values for categorical features	Any	Boolean	True	Indicate whether unknown values of existing categorical features can be mapped to a new, additional feature

3. **Decision Jungle**
   The **Two-Class Decision Jungle** module returns an untrained classifier. You then train this model on a labeled training data set, by using Train Model or Tune Model Hyperparameters. The trained model can then be used to make predictions.

Module parameters

Name	Range	Type	Default	Description
Resampling method	Any	ResamplingMethod	Bagging	Choose a resampling method
Number of decision DAGs	>=1	Integer	8	Specify the number of decision graphs to build in the ensemble
Maximum depth of the decision DAGs	>=1	Integer	32	Specify the maximum depth of the decision graphs in the ensemble
Maximum width of the decision DAGs	>=8	Integer	128	Specify the maximum width of the decision graphs in the ensemble
Number of optimization steps per decision DAG layer	>=1000	Integer	2048	Specify the number of steps to use to optimize each level of the decision graphs
Allow unknown values for categorical features	Any	Boolean	True	Indicate whether unknown values of existing categorical features can be mapped to a new, additional feature

4. **Boosted Decision Tree**

   A boosted decision tree is an ensemble learning method in which the second tree corrects for the errors of the first tree, the third tree corrects for the errors of the first and second trees, and so forth. Predictions are based on the entire ensemble of trees together that makes the prediction.

Module parameters

Name	Range	Type	Default	Description
Maximum number of leaves per tree	>=1	Integer	20	Specify the maximum number of leaves allowed per tree
Minimum number of samples per leaf node	>=1	Integer	10	Specify the minimum number of cases required to form a leaf
Learning rate	[double.Epsilon;1.0]	Float	0.2	Specify the initial learning rate
Number of trees constructed	>=1	Integer	100	Specify the maximum number of trees that can be created during training
Random number seed	Any	Integer		Type a value to seed the random number generator that is used by the model. Leave it blank for the default.
Allow unknown categorical levels	Any	Boolean	True	If True, an additional level is created for each categorical column. Any levels in the test dataset that are not available in the training dataset are mapped to this additional level.

5. Neural Network

A neural network is a set of interconnected layers. The inputs are the first layer, and are connected to an output layer by an acyclic graph comprised of weighted edges and nodes.

Between the input and output layers you can insert multiple hidden layers. Most predictive tasks can be accomplished easily with only one or a few hidden layers. However, recent research has shown that deep neural networks (DNN) with many layers can be very effective in complex tasks such as image or speech recognition. The successive layers are used to model increasing levels of semantic depth.

Module parameters

Name	Range	Type	Default	Description
Hidden layer specification	List	Neural Network Topology	Fully-connected case	Specify the architecture of the hidden layer or layers
The initial learning weights diameter	>=double.Epsilon	Float	0.1	Specify the node weights at the start of the learning process
Learning rate	[double.Epsilon;1.0]	Float	0.1	Specify the size of each step in the learning process
The momentum	[0.0;1.0]	Float	0.0	Specify a weight to apply during learning to nodes from previous iterations
Neural network definition	Any	StreamReader		When you select **Custom definition script**, type a valid script expression on each line to define the layers, nodes, and behavior of a custom neural network
The type of normalizer	List	Normalization Method	Min-Max normalizer	Select the type of normalization to apply to learning examples
Number of learning iterations	>=1	Integer	100	Specify the number of iterations performed during learning
Shuffle examples	Any	Boolean	true	Select this option to change the order of instances between learning iterations
Random number seed	Any	Integer		Specify a numeric seed to use for random number generation. Leave it blank to use the default seed.
Allow unknown categorical levels	Any	Boolean	True	Indicated whether an additional level should be created for unknown categories. If the test dataset contains categories that are not present in the training dataset, they are mapped to this unknown level.

6. Averaged Perceptron

The *averaged perceptron method* is an early and very simple version of a neural network. In this approach, inputs are classified into several possible outputs based on a linear function, and then combined with a set of

weights that are derived from the feature vector—hence the name "perceptron."

Module parameters

Name	Range	Type	Default	Description
Learning rate	>=double.Epsilon	Float	1.0	The initial learning rate for the Stochastic Gradient Descent optimizer.
Maximum number of iterations	>=1	Integer	10	The number of Stochastic Gradient Descent iterations to be performed over the training dataset.
Random number seed	Any	Integer		The seed for the random number generator used by the model. Leave it blank for the default.
Allow unknown categorical levels	Any	Boolean	True	If True, creates an additional level for each categorical column. Any levels in the test dataset that are not available in the training dataset are mapped to this additional level.

7. Support Vector Machine

Support vector machines are among the earliest of machine learning algorithms, and SVM models have been used in many applications, from information retrieval to text and image classification. SVMs can be used for both classification and regression tasks.

Module parameters

Name	Range	Type	Default	Description
Number of iterations	>=1	Integer	1	The number of iterations
Lambda	>=double.Epsilon	Float	0.001	Weight for L1 regularization. Using a non-zero value avoids overfitting the model to the training dataset.
Normalize features	Any	Boolean	True	If True, normalize the features.
Project to the unit-sphere	Any	Boolean	False	If True, project the features to a unit circle.
Random number seed	Any	Integer		The seed for the random number generator used by the model. Leave it blank for the default.
Allow unknown categorical levels	Any	Boolean	True	If True, creates an additional level for each categorical column. Any levels in the test dataset that are not available in the training dataset are mapped to this additional level.

8. Locally Deep SVM

Support vector machines (SVMs) are an extremely popular and well-researched class of supervised learning models, which can be used in linear and non-linear classification tasks. Recent research has focused on ways to optimize these models to efficiently scale to larger training sets. In this

implementation from Microsoft Research, the kernel function that is used for mapping data points to feature space is specifically designed to reduce the time needed for training while maintaining most of the classification accuracy.

Name	Range	Type	Default	Description
Create trainer mode	List	Learner parameter option	Single Parameter	Advanced learner options: 1. Create learner using a single parameter 2. Create learner using a parameter range
Depth of the tree	>=1	Integer	3	The depth of the locally deep SVM tree.
Lambda W	>=1.401298E-45	Float	0.1	Regularization weight for the classifier parameter Lambda W.
Lambda Theta	>=1.401298E-45	Float	0.01	Regularization weight for the classifier parameter Lambda Theta.
Lambda Theta Prime	>=1.401298E-45	Float	0.01	Regularization weight for the classifier parameter Lambda Theta prime.
Sigmoid sharpness	>=1.401298E-45	Float	1.0	The sigmoid sharpness.
Depth of the tree	[1;int.MaxValue]	ParameterRangeSettings	1; 3; 5; 7	The range for the depth of the locally deep SVM tree.
Lambda W	[1.401298E-45;3.40282347E+38]	ParameterRangeSettings	0.1; 0.01; 0.001	Range for the regularization weight for the classifier parameter Lambda W.
Lambda Theta	[1.401298E-45;3.40282347E+38]	ParameterRangeSettings	0.1; 0.01; 0.001	Range for the regularization weight for the classifier parameter Lambda Theta.
Lambda Theta Prime	[1.401298E-45;3.40282347E+38]	ParameterRangeSettings	0.1; 0.01; 0.001	Range for the regularization weight for the classifier parameter Lambda Theta prime'.
Sigmoid sharpness	[1.401298E-45;3.40282347E+38]	ParameterRangeSettings	1.0; 0.1; 0.01	The range for the sigmoid sharpness.
Feature normalizer	List	Normalizer type	Min-Max normalizer	The type of normalization to apply to learning examples.
Number of iterations	>=1	Integer	15000	Number of learning iterations.
Number of iterations	[1;int.MaxValue]	ParameterRangeSettings	10000; 15000; 20000	The range for the number of learning iterations.
Random number seed	Any	Integer		The seed for the random number generator that is used by the model. Leave it blank for the default.
Allow unknown categorical levels	Any	Boolean	True	If True, creates an additional level for each categorical column. Any levels in the test dataset that are not available in the training dataset are mapped to this additional level.

- **Multi-class classification**

1. **Logistic Regression**

   Classification using logistic regression is a supervised learning method, and therefore requires a labeled dataset. You train the model by providing the model and the labeled dataset as an input to a module such as Train Model or Tune Model Hyperparameters. The trained model can then be used to predict values for new input examples.

Module parameters

Name	Range	Type	Default	Description
Optimization tolerance	>=double.Epsilon	Float	0.0000001	Specify a tolerance value for the L-BFGS optimizer
L1 regularization weight	>=0.0	Float	1.0	Specify the L1 regularization weight. Use a non-zero value to avoid overfitting.
L2 regularization weight	>=0.0	Float	1.0	Specify the L2 regularization weight. Use a non-zero value to avoid overfitting.
Memory size for L-BFGS	>=1	Integer	20	Specify the amount of memory (in MB) to use for the L-BFGS optimizer. When less memory is used, training is faster, but less accurate.
Random number seed	Any	Integer		Type a value to seed the random number generator used by the model. Leave it blank for the default.
Allow unknown categorical levels	Any	Boolean	True	Indicate whether an additional level should be created for each categorical column. Any levels in the test dataset that are not available in the training dataset are mapped to this additional level.

2. **Decision Forest**

   The decision forest algorithm is an ensemble learning method for classification. The algorithm works by building multiple decision trees and then *voting* on the most popular output class. Voting is a form of aggregation, in which each tree in a classification decision forest outputs a non-normalized frequency histogram of labels. The aggregation process sums these histograms and normalizes the result to get the "probabilities" for each label. The trees that have high prediction confidence have a greater weight in the final decision of the ensemble.

Module parameters

Name	Range	Type	Default	Description
Resampling method	Any	ResamplingMethod	Bagging	Choose a resampling method: **Bagging** or **Replicate**
Number of decision trees	>=1	Integer	8	Specify the number of decision trees to create in the ensemble
Maximum depth of the decision trees	>=1	Integer	32	Specify the maximum depth of any decision tree that can be created
Number of random splits per node	>=1	Integer	128	Specify the number of splits generated per node, from which the optimal split is selected
Minimum number of samples per leaf node	>=1	Integer	1	Specify the minimum number of training samples required to generate a leaf node
Allow unknown values for categorical features	Any	Boolean	True	Indicate whether unknown values of existing categorical features can be mapped to a new, additional feature

3. **Decision Jungle**

   You define the model and its parameters using this module, and then connect a labeled training data set to train the model using one of the training modules. The trained model can be used to predict a target that has multiple values.

Module parameters

Name	Range	Type	Default	Description
Resampling method	Any	ResamplingMethod	Bagging	Choose a resampling method
Number of decision DAGs	>=1	Integer	8	Specify the number of decision graphs that can be created in the ensemble
Maximum depth of the decision DAGs	>=1	Integer	32	Specify the maximum depth of the decision graphs to create in the ensemble
Maximum width of the decision DAGs	>=8	Integer	128	Specify the maximum width of the decision graphs to create in the ensemble
Number of optimization steps per decision DAG layer	>=1000	Integer	2048	Specify the number of steps to use to optimize each level of the decision graphs
Allow unknown values for categorical features	Any	Boolean	True	Indicate whether unknown values of existing categorical features can be mapped to a new, additional feature

4. **Neural Network**

   For example, neural networks of this kind might be used in complex computer vision tasks, such as digit or letter recognition, document classification, and pattern recognition.

Classification using neural networks is a supervised learning method, and therefore requires a *tagged dataset* that includes a label column.

Module parameters

Name	Range	Type	Default	Description
Hidden layer specification	List	Neural Network Topology	Fully-connected case	Specify the architecture of the hidden layer or layers
The initial learning weights diameter	>=double.Epsilon	Float	0.1	Specify the node weights at the start of the learning process
The learning rate	[double.Epsilon;1.0]	Float	0.1	Specify the size of each step in the learning process
The momentum	[0.0;1.0]	Float	0.0	Specify a weight to apply during learning to nodes from previous iterations
Neural network definition	Any	StreamReader		When you select **Custom definition script**, type a valid script expression on each line to define the layers, nodes, and behavior of a custom neural network
The type of normalizer	List	Normalization Method	Minimum-maximum normalizer	Select the type of normalization to apply to learning examples
Number of learning iterations	>=1	Integer	100	Specify the number of iterations while learning
Shuffle examples	Any	Boolean	True	Select this option to change the order of instances between learning iterations
Random number seed	Any	Integer		Specify a numeric seed to use for random number generation. Leave blank to use the default seed.
Allow unknown categorical levels	Any	Boolean	True	Indicate whether an additional level should be created for unknown categories. If the test dataset contains categories that are not present in the training dataset, they are mapped to this unknown level.

- # Regression

1. Linear Regression

The **Linear Regression** module in Machine Learning Studio (classic), to create a linear regression model for use in an experiment. Linear regression attempts to establish a linear relationship between one or more independent variables and a numeric outcome, or dependent variable.

Module parameters

Name	Range	Type	Default	Description
Normalize features	any	Boolean	true	Indicate whether instances should be normalized
Average final hypothesis	any	Boolean	true	Indicate whether the final hypothesis should be averaged
Learning rate	>=double.Epsilon	Float	0.1	Specify the initial learning rate for the stochastic gradient descent optimizer
Number of training epochs	>=0	Integer	10	Specify how many times the algorithm should iterate through examples. For datasets with a small number of examples, this number should be large to reach convergence.
Decrease learning rate	Any	Boolean	true	Indicate whether the learning rate should decrease as iterations progress
L2 regularization weight	>=0.0	Float	0.001	Specify the weight for L2 regularization. Use a non-zero value to avoid overfitting.
Random number seed	any	Integer		Specify a value to seed the random number generator used by the model. Leave blank for default.
Allow unknown categorical levels	any	Boolean	true	Indicate whether an additional level should be created for each categorical column. Any levels in the test dataset not available in the training dataset are mapped to this additional level.
Include intercept term	Any	Boolean	True	Indicate whether an additional term should be added for the intercept

2. Bayesian Linear

The Bayesian approach uses linear regression supplemented by additional information in the form of a prior probability distribution. Prior information about the parameters is combined with a likelihood function to generate estimates for the parameters.

Module parameters

Name	Range	Type	Default	Description
Regularization weight	>=double.Epsilon	Float	1.0	Type a constant to use in regularization. The constant represents the ratio of the precision of weight prior to the precision of noise.
Allow unknown categorical levels	Any	Boolean	true	If true creates an additional level for each categorical column. Any levels in the test dataset not available in the training dataset are mapped to this additional level.

3. Boosted Decision Tree

Boosted Decision Tree Regression module in Machine Learning Studio (classic), to create an ensemble of regression trees using boosting. *Boosting* means that each tree is dependent on prior trees. The algorithm learns by fitting the residual of the trees that preceded it. Thus, boosting in a decision tree ensemble tends to improve accuracy with some small risk of less coverage.

Module parameters

Name	Range	Type	Default	Description
Maximum number of leaves per tree	>=1	Integer	20	Specify the maximum number of leaves per tree
Minimum number of samples per leaf node	>=1	Integer	10	Specify the minimum number of cases required to form a leaf node
Learning rate	[double.Epsilon;1.0]	Float	0.2	Specify the initial learning rate
Total number of trees constructed	>=1	Integer	100	Specify the maximum number of trees that can be created during training
Random number seed	any	Integer		Provide a seed for the random number generator used by the model. Leave blank for default.
Allow unknown categorical levels	any	Boolean	true	If true, create an additional level for each categorical column. Levels in the test dataset not available in the training dataset are mapped to this additional level.

4. Fast Forest Quantile

Fast Forest Quantile Regression module in Machine Learning Studio (classic), to create a regression model that can predict values for a specified number of quantiles.

Quantile regression is useful if you want to understand more about the distribution of the predicted value, rather than get a single mean prediction value. This method has many applications, including:

- Predicting prices
- Estimating student performance or applying growth charts to assess child development
- Discovering predictive relationships in cases where there is only a weak relationship between variables

Module parameters

Name	Type	Range	Optional	Description	Default
Create trainer mode	CreateLearnerMode	List:Single Parameter|Parameter Range	Required	Single Parameter	Create advanced learner options
Number of Trees	Integer		mode:Single Parameter	100	Specify the number of trees to be constructed
Number of Leaves	Integer		mode:Single Parameter	20	Specify the maximum number of leaves per tree. The default number is 20
Minimum number of training instances required to form a leaf	Integer		mode:Single Parameter	10	Indicates the minimum number of training instances required to form a leaf
Bagging fraction	Float		mode:Single Parameter	0.7	Specifies the fraction of training data to use for each tree
Feature fraction	Float		mode:Single Parameter	0.7	Specifies the fraction of features (chosen randomly) to use for each tree
Split fraction	Float		mode:Single Parameter	0.7	Specifies the fraction of features (chosen randomly) to use for each split
Quantile sample count	Integer	Max: 2147483647	mode:Single Parameter	100	Specifies number of instances used in each node to estimate quantiles
Quantiles to be estimated	String		mode:Single Parameter	"0.25;0.5;0.75"	Specifies the quantile to be estimated

Random number seed	Integer		Optional		Provide a seed for the random number generator used by the model. Leave blank for default.
Allow unknown categorical levels	Boolean		Required	true	If true, create an additional level for each categorical column. Levels in the test dataset not available in the training dataset are mapped to this additional level.
Maximum number of leaves per tree	ParameterRangeSettings	[16;128]	mode:Parameter Range	16; 32; 64	Specify range for the maximum number of leaves allowed per tree
Number of trees constructed	ParameterRangeSettings	[1;256]	mode:Parameter Range	16; 32; 64	Specify the range for the maximum number of trees that can be created during training
Minimum number of samples per leaf node	ParameterRangeSettings	[1;10]	mode:Parameter Range	1; 5; 10	Specify the range for the minimum number of cases required to form a leaf
Range for bagging fraction	ParameterRangeSettings	[0.25;1.0]	mode:Parameter Range	0.25; 0.5; 0.75	Specifies the range for fraction of training data to use for each tree
Range for feature fraction	ParameterRangeSettings	[0.25;1.0]	mode:Parameter Range	0.25; 0.5; 0.75	Specifies the range for fraction of features (chosen randomly) to use for each tree
Range for split fraction	ParameterRangeSettings	[0.25;1.0]	mode:Parameter Range	0.25; 0.5; 0.75	Specifies the range for fraction of features (chosen randomly) to use for each split

5. Neural Network Regression

Although neural networks are widely known for use in deep learning and modeling complex problems such as image recognition, they are easily adapted to regression problems. Any class of statistical models can be termed a neural network if they use adaptive weights and can approximate non-linear functions of their inputs. Thus neural network regression is suited to problems where a more traditional regression model cannot fit a solution.

Module parameters

Name	Range	Type	Default	Description
Hidden layer specification	List	Neural Network Topology	Fully-connected case	Specify the architecture of the hidden layer or layers
The initial learning weights diameter	>=double.Epsilon	Float	0.1	Specify the node weights at the start of the learning process
Learning rate	[double.Epsilon;0.01]	Float	0.005	Specify the size of each step in the learning process
The momentum	[0.0;1.0]	Float	0.0	Specify a weight to apply during learning to nodes from previous iterations
Neural network definition	Any	StreamReader		When you select "Custom definition script", type a valid script expression on each line to define the layers, nodes, and behavior of a custom neural network
The type of normalizer	List	Normalization Method	Min-Max normalizer	Select the type of normalization to apply to learning examples
Number of hidden nodes	Any	String	100	Type the number of nodes in the hidden layer. For multiple hidden layers, type a comma-separated list.
Number of learning iterations	>=1	Integer	100	Specify the number of iterations while learning
Shuffle examples	Any	Boolean	true	Select this option to change the order of instances between learning iterations
Random number seed	Any	Integer		Specify a numeric seed to use for random number generation. Leave blank to use the default seed. This parameter is optional
Allow unknown categorical levels	Any	Boolean	true	Indicate whether an additional level should be created for unknown categories. If the test dataset contains categories not present in the training dataset they are mapped to this unknown level.

6. Ordinal Regression

Ordinal Regression module in Machine Learning Studio (classic), to create a regression model that can be used to predict ranked values.

Some examples of ranked values:

- Survey responses that capture user's preferred brands on a 1 to 5 scale
- The order of finishers in a race
- URLs in ranked search results

This module solves a ranking problem as a series of related classification problems. Therefore, the algorithm creates a series of extended training examples using a binary model for each rank, and trains against that extended set. This operation can be computationally expensive.

1. Add the **Ordinal Regression Model** module to your experiment in Studio (classic). You can find this module under **Machine Learning - Initialize**, in the **Regression** category.
2. Add a module that supports binary classification, and configure the model. There are several two-class modules in the classification category.

3. Connect the binary classification model as an input to the **Ordinal Regression Model** module.
4. Additional parameters are not required on the **Ordinal Regression Model**; the algorithm has been pre-configured with the most effective parameters for solving a ranking problem.
5. Connect a training dataset and the Train Model module.
6. In the Train Model module, select the column that contains the rank values.

   The rank values must be numerical values, but they need not be integers or positive numbers, as long as they represent a sequence.

   For purposes of processing, the ranks are assumed to have the order 1 to K, where 1 is lowest rank, and K is the highest rank. However, the Train Model module can work even if the semantics of your scale are reversed.

   For example, if in your original survey, 1 was the highest score and 5 is the lowest, it does not affect the processing of the model.

7. Run the experiment.

- # Anomaly Detection

1. **SVM**

- Support vector machines (SVMs) are supervised learning models that analyze data and recognize patterns, and that can be used for both classification and regression tasks.
- Typically, the SVM algorithm is given a set of training examples labeled as belonging to one of two classes. An SVM model is based on dividing the training sample points into separate categories by as wide a gap as possible, while penalizing training samples that fall on the wrong side of the gap. The SVM model then makes predictions by assigning points to one side of the gap or the other.

Module parameters

Name	Type	Range	Optional	Description	Default
Create trainer mode	Create Trainer Mode	List:Single Parameter|Parameter Range	Required	Single Parameter	Specify learner options. Use the **SingleParameter** option to manually specify all values. Use the **ParameterRange** option to sweep over tunable parameters.
nu	Float	>=double.Epsilon	mode:Single Parameter	0.1	This parameter (represented by the Greek letter nu) determines the trade-off between the fraction of outliers and the number of support vectors.
epsilon	Float	>=double.Epsilon	mode:Single Parameter	0.001	Specifies the stopping tolerance.
psnu	ParameterRangeSettings	[0.001;1.0]	mode:Parameter Range	0.001; 0.01; 0.1	Specifies the range for the trade-off between the fraction of outliers and the number of support vectors.
psEpsilon	ParameterRangeSettings	[1e-6;1.0]	mode:Parameter Range	0.001; 0.01; 0.1	Specifies the range for stopping tolerance.

ONE V/S ONE MULTICLASS

Some classification algorithms permit the use of more than two classes by design. Others restrict the possible outcomes to one of two values (a binary, or two-class model). But even binary classification algorithms can be adapted for multi-class classification tasks through a variety of strategies.

This component implements the one-versus-one method, in which a binary model is created per class pair. At prediction time, the class which received the most votes is selected. Since it requires to fit `n_classes * (n_classes - 1) / 2` classifiers, this method is usually slower than one-versus-all, due to its O(n_classes^2) complexity. However, this method may be advantageous for algorithms such as kernel algorithms which don't scale well with `n_samples`. This is because each individual learning problem only involves a small subset of the data whereas, with one-versus-all, the complete dataset is used `n_classes` times.

In essence, the component creates an ensemble of individual models and then merges the results, to create a single model that predicts all classes. Any binary classifier can be used as the basis for a one-versus-one model.

For example, let's say you configure a Two-Class Support Vector Machine model and provide that as input to the One-vs-One Multiclass component. The component would create two-class support vector machine models for all members of the output class. It would then apply the one-versus-one method to combine the results for all classes.

This component creates an ensemble of binary classification models to analyze multiple classes. To use this component, you need to configure and train a *binary classification* model first.

You connect the binary model to the One-vs-One Multiclass component. You then train the ensemble of models by using Train Model with a labeled training dataset.

When you combine the models, One-vs-One Multiclass creates multiple binary classification models, optimizes the algorithm for each class, and then merges the models. The component does these tasks even though the training dataset might have multiple class values.

1. Add the One-vs-One Multiclass component to your pipeline in the designer. You can find this component under **Machine Learning - Initialize**, in the **Classification** category.

   The One-vs-One Multiclass classifier has no configurable parameters of its own. Any customizations must be done in the binary classification model that's provided as input.

2. Add a binary classification model to the pipeline, and configure that model. For example, you might use Two-Class Support Vector Machine or Two-Class Boosted Decision Tree.
3. Add the Train Model component to your pipeline. Connect the untrained classifier that is the output of One-vs-One Multiclass.
4. On the other input of Train Model, connect a labeled training dataset that has multiple class values.
5. Submit the pipeline.

   Result-

   1. After training is complete, you can use the model to make multiclass predictions.
   2. Alternatively, you can pass the untrained classifier to Cross-Validate Model for cross-validation against a labeled validation dataset.

ONE V/S ALL MULTICLASS

Some classification algorithms permit the use of more than two classes by design. Others restrict the possible outcomes to one of two values (a binary, or two-class

model). But even binary classification algorithms can be adapted for multi-class classification tasks through a variety of strategies.

This component implements the one-versus-all method, in which a binary model is created for each of the multiple output classes. The component assesses each of these binary models for the individual classes against its complement (all other classes in the model) as though it's a binary classification issue. In addition to its computational efficiency (only `n_classes` classifiers are needed), one advantage of this approach is its interpretability. Since each class is represented by one and one classifier only, it is possible to gain knowledge about the class by inspecting its corresponding classifier. This is the most commonly used strategy for multiclass classification and is a fair default choice. The component then performs prediction by running these binary classifiers and choosing the prediction with the highest confidence score.

In essence, the component creates an ensemble of individual models and then merges the results, to create a single model that predicts all classes. Any binary classifier can be used as the basis for a one-versus-all model.

For example, let's say you configure a Two-Class Support Vector Machine model and provide that as input to the One-vs-All Multiclass component. The component would create two-class support vector machine models for all members of the output class. It would then apply the one-versus-all method to combine the results for all classes.

This component creates an ensemble of binary classification models to analyze multiple classes. To use this component, you need to configure and train a *binary classification* model first.

You connect the binary model to the One-vs-All Multiclass component. You then train the ensemble of models by using Train Model with a labeled training dataset.

When you combine the models, One-vs-All Multiclass creates multiple binary classification models, optimizes the algorithm for each class, and then merges the models. The component does these tasks even though the training dataset might have multiple class values.

1. Add the One-vs-All Multiclass component to your pipeline in the designer. You can find this component under **Machine Learning - Initialize**, in the **Classification** category.

The One-vs-All Multiclass classifier has no configurable parameters of its own. Any customizations must be done in the binary classification model that's provided as input.

2. Add a binary classification model to the pipeline, and configure that model. For example, you might use Two-Class Support Vector Machine or Two-Class Boosted Decision Tree.
3. Add the Train Model component to your pipeline. Connect the untrained classifier that is the output of One-vs-All Multiclass.
4. On the other input of Train Model, connect a labeled training dataset that has multiple class values.
5. Submit the pipeline.

Results

1. After training is complete, you can use the model to make multiclass predictions.
2. Alternatively, you can pass the untrained classifier to Cross-Validate Model for cross-validation against a labeled validation dataset.