**Classification modules**

**Two-Class Locally Deep Support Vector Machine**

To create a two-class, non-linear support vector machines (SVM) classifier that is optimized for efficient prediction. 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. This model is a supervised learning method, and therefore requires a tagged dataset, which includes a label column. After you define the model parameters, train it by providing the model and a tagged dataset as input to Train Model or Tune Model Hyperparameters. The trained model can then be used to predict values for new inputs.

**How to configure Two-Class Locally Deep Support Vector Machine**

1. Add the Two-Class Locally-Deep Support Vector Machine module to your experiment in Studio (classic).
2. Specify how you want the model to be trained, by setting the Create trainer mode option.

* Single Parameter: If you know how you want to configure the model, provide a specific set of values as arguments.
* Parameter Range: If you are not sure of the best parameters, you can find the optimal parameters by specifying multiple values and using the Tune Model Hyperparameters module to find the optimal configuration. The

trainer iterates over multiple combinations of the settings you provided and determines the combination of values that produces the best model.

1. For Depth of the tree, specify the maximum depth of the tree that can be created by the local deep kernel learning SVM (LD-SVM) model.

The cost of training increases linearly with tree depth; therefore, choose an appropriate depth, depending on how much time you can afford to spend when building the model. Training time should roughly double as the depth is increased by one. Prediction accuracy should increase, reach a peak, and then decrease with increasing depth.

1. For Lambda W, specify the weight that should be given to the regularization term.

Regularization restricts large value components in the trained classifier. When the number of samples is insufficient given the number of features, you can use L2 regularization to avoid overfitting. Larger values for Lambda W mean that more emphasis is placed on regularizing the classifier weights and less on the training-set classification error. If the default value (0.1) doesn’t work well, you should also try {0.0001, 0.001 and 0.01}.

1. For Lambda Theta, specify how much space should be left between a region boundary and the closest data point.

This model works by partitioning the data space and feature space into regions. When Lambda Theta is minimized in such a way that region boundaries in the trained model are too close to the training data points, the model might yield a low training error, but a high test error, due to overfitting.To reduce the number of parameters that need to be validated, a good rule of thumb is to set Lambda Theta to one-tenth of the value that is used for Lambda W. Larger values mean that more emphasis is put on preventing overfitting than on minimizing classification errors in the training set. If the default value (0.01) doesn’t work well, you should also try {0.0001, 0.001 and 0.1}.

1. For Lambda Theta Prime, type a value to control the amount of curvature that is allowed in decision boundaries in the model.

Larger values give the model the flexibility to learn curved decision boundaries, while smaller values might constrain the decision boundaries to more of a stepwise linear pattern.

This parameter works in conjunction with the Sigma parameter. To reduce the number of parameters that need to be validated, a good rule of thumb is to set Lambda Theta Prime to one-tenth the value of Lambda W. If the default value (0.01) doesn’t work well, you should also try {0.0001, 0.001 and 0.1,}.

1. For Sigmoid sharpness, type a value to use for the scaling parameter σ.

Larger values mean that the tanh in local kernel Θ (theta) is saturated, whereas a smaller value implies a more linear operating range for theta.

If the default value (1) does not work well, you can also try {0.1, 0.01, 0.001}.

1. In Number of iterations, indicate how many times the algorithm should update the classifier parameters with a random subset of examples.
2. For Feature normalizer, choose a method to use in normalizing feature values. The following methods are supported:

* ***Binning normalizer:*** The binning normalizer creates bins of equal size, and then normalizes every value in each bin to be divided by the total number of bins.
* ***Min-Max normalizer:*** The min-max normalizer linearly rescales every feature to the [0,1] interval. Rescaling to the [0,1] interval is done by shifting the values of each feature so that the minimal value is 0, and then dividing by the new maximal value (which is the difference between the original maximal and minimal values).
* ***Gaussian normalizer:*** The Gaussian normalizer rescales the values of each feature to have a mean of 0 and a variance of 1. This is done by computing the mean and the variance of each feature. Then, for each instance, the mean value is subtracted, and the result divided by the square root of the variance (the standard deviation).
* ***Do not normalize:*** No normalization is performed.

1. In Random number seed, type a value to use as a seed if you want to ensure reproducibility across runs.
2. Select the Allow unknown categorical levels option to create a group for unknown values in the testing or validation sets. If you deselect it, the model can accept only the values that are contained in the training data. In the former case, the model might be less precise for known values, but it can provide better predictions for new (unknown) values.
3. Connect a tagged dataset and one of the training modules:

* If you set Create trainer mode to Single Parameter, use the Train Model module.
* If you set Create trainer mode to Parameter Range, use the Tune Model Hyperparameters module.

1. Run the experiment.

**Results**

After training is complete:

* To see a summary of the model's parameters, right-click the output of the Train Model module or Tune Model Hyperparameters module, and select Visualize.
* To save a snapshpt of the trained model, right-click the Trained model output and select Save As Trained Model. This model is not updated on successive runs of the same experiment.
* To perform cross-validation against a labeled data set, connect the untrained model to Cross-Validate Model.

**Tips**

This LD-SVM classifier is most useful under the following conditions:

* You have a binary classification issue, or you can reduce your issue to a binary classification task.
* You tried a linear classifier, but it did not perform well.
* You tried a non-linear SVM or other classifier, and got good classification accuracy, but it took too long to train the model.
* You can afford to sacrifice prediction accuracy to reduce training time.

LD-SVM models are a good choice when your data is complicated enough that linear models (such as logistic regression) perform poorly. LD-SVM models are also small enough to be used in mobile devices or other scenarios where complex models (such as neural networks) are too big to be consumed efficiently.

Conversely, this model should not be used if you don’t care about model size or if a linear model is required for simplicity or prediction speed. There is also no point to changing to LD-SVM if linear classifiers are already giving good results,

or if you can get high classification accuracy by adding small amounts of non-linearity.

**Implementation details**

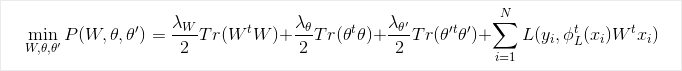
The LD-SVM model was developed by Microsoft Research as part of ongoing efforts to speed up non-linear SVM prediction. The work of Gonen and Alpaydin (2008) on the localized multiple kernel learning method was particularly valuable. Use of a local kernel function enables the model to learn arbitrary local feature embeddings, including high-dimensional, sparse, and computationally deep features that introduce non-linearities into the model.

LD-SVM is faster than most other classifiers for several reasons:

* The model learns decision boundaries that are locally linear. Therefore, a test point can be efficiently classified by testing it against its local decision boundary, rather than testing against the entire set of decision boundaries all over feature space.
* The model uses efficient primal-based routines to optimize the space of tree-structured local feature embeddings that scale to large training sets with more than half a million training points.
* The cost of testing a point against its local decision boundary is logarithmic in the number of training points.

As a consequence of these optimizations, training the LD-SVM model is exponentially faster than training traditional SVM models.

**Optimization formula**



**Module parameters**

| **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] | Parameter Range  Settings | 1; 3; 5; 7 | The range for the depth of the locally deep SVM tree. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Lambda W | [1.401298E-45;3.40282347E+38] | Parameter Range  Settings | 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] | Parameter Range  Settings | 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] | Parameter Range  Settings | 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] | Parameter Range Settings | 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] | Parameter Range  Settings | 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. |

**Two-Class Logistic Regression**

To create a logistic regression model that can be used to predict two (and only two) outcomes. 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.

**Logistic regression**

Logistic regression is a well-known method in statistics that is used to predict the probability of an outcome, and is especially popular for classification tasks. The algorithm predicts the probability of occurrence of an event by fitting data to a logistic function.In this module, the classification algorithm is optimized for dichotomous or binary variables. if you need to classify multiple outcomes, use the Multiclass Logistic Regression module.

**How to configure Two-Class Logistic Regression**

To train this model, you must provide a dataset that contains a label or class column. Because this module is intended for two-class problems, the label or class column must contain exactly two values.

For example, the label column might be [Voted] with possible values of "Yes" or "No". Or, it might be [Credit Risk], with possible values of "High" or "Low".

1. Add the Two-Class Logistic Regression module to your experiment in Studio (classic).
2. Specify how you want the model to be trained, by setting the Create trainer mode option.

* Single Parameter: If you know how you want to configure the model, you can provide a specific set of values as arguments.
* Parameter Range: If you are not sure of the best parameters, you can find the optimal parameters by specifying multiple values and using the Tune Model Hyperparameters module to find the optimal configuration. The trainer iterates over multiple combinations of the settings and determines the combination of values that produces the best model.

1. For Optimization tolerance, specify a threshold value to use when optimizing the model. If the improvement between iterations falls below the specified threshold, the algorithm is considered to have converged on a solution, and training stops.
2. For L1 regularization weight and L2 regularization weight, type a value to use for the regularization parameters L1 and L2. A non-zero value is recommended for both.

***Regularization*** is a method for preventing overfitting by penalizing models with extreme coefficient values. Regularization works by adding the penalty that is associated with coefficient values to the error of the hypothesis. Thus, an accurate model with extreme coefficient values would be penalized more, but a less accurate model with more conservative

values would be penalized less.

L1 and L2 regularization have different effects and uses.

* L1 can be applied to sparse models, which is useful when working with high-dimensional data.
* In contrast, L2 regularization is preferable for data that is not sparse.

This algorithm supports a linear combination of L1 and L2 regularization values: that is, if x = L1 and y = L2, then ax + by = c defines the linear span of the regularization terms.

1. For Memory size for L-BFGS, specify the amount of memory to use for L-BFGS optimization.

L-BFGS stands for "limited memory Broyden-Fletcher-Goldfarb-Shanno". It is an optimization algorithm that is popular for parameter estimation. This parameter indicates the number of past positions and gradients to store for the computation of the next step.

This optimization parameter limits the amount of memory that is used to compute the next step and direction. When you specify less memory, training is faster but less accurate.

1. For Random number seed, type an integer value. Defining a seed value is important if you want the results to be reproducible over multiple runs of the same experiment.
2. Select the Allow unknown categorical levels option to create an additional “unknown” level in each categorical column. If you do so, any values (levels) in the test dataset that are not available in the training dataset are mapped to this "unknown" level.
3. Add a tagged dataset to the experiment, and connect one of the training modules.

* If you set Create trainer mode to Single Parameter, use the Train Model module.
* If you set Create trainer mode to Parameter Range, use the Tune Model Hyperparameters module.

1. Run the experiment.

**Results**

After training is complete:

* To see a summary of the model's parameters, together with the feature weights learned from training, right-click the output of Train Model or Tune Model Hyperparameters, and select Visualize.
* To make predictions on new data, use the trained model and new data as input to the Score Model module.
* To perform cross-validation against a labeled data set, connect the data and the untrained model to Cross-Validate Model.

**Implementation details**

Logistic regression assumes a *logistic distribution* of the data, where the probability that an example belongs to class 1 is the formula:

p(x;β0,…, βD-1)

Where:

* x is a D-dimensional vector containing the values of all the features of the instance.
* p is the logistic distribution function.
* β{0},..., β {D-1} are the unknown parameters of the logistic distribution.

The algorithm tries to find the optimal values for β{0},..., β {D-1} by maximizing the log probability of the parameters given the inputs.

**Two-Class Neural Network**

To create a neural network model that can be used to predict a target that has only two values. Classification using neural networks is a supervised learning method, and therefore requires a tagged dataset, which includes a label column. For example, you could use this neural network model to predict binary outcomes such as whether or not a patient has a certain disease, or whether a machine is likely to fail within a specified window of time. After you define the model, train it by providing a tagged dataset and the model as an input to Train Model or to Tune Model Hyperparameters. The trained model can then be used to predict values for new inputs.

**Neural networks**

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. The relationship between inputs and outputs is learned from training the neural network on the input data. The direction of the graph

proceeds from the inputs through the hidden layer and to the output layer. All nodes in a layer are connected by the weighted edges to nodes in the next layer.

To compute the output of the network for a particular input, a value is calculated at each node in the hidden layers and in the output layer. The value is set by calculating the weighted sum of the values of the nodes from the previous layer. An activation function is then applied to that weighted sum.

**How to configure Two-Class Neural Network**

1. Add the Two-Class Neural Network module to your experiment in Studio (classic). You can find this module under Machine Learning, Initialize, in the Classification category.
2. Specify how you want the model to be trained, by setting the Create trainer mode option.

* Single Parameter: Choose this option if you already know how you want to configure the model.
* Parameter Range: Choose this option if you are not sure of the best parameters. Then, specify a range of values and use the Tune Model Hyperparameters module to iterate over the combinations and find the optimal configuration.

1. For Hidden layer specification, select the type of network architecture to create.

* ***Fully-connected case:*** Uses the default neural network architecture, defined for two-class neural networks as follows:
* Has one hidden layer.
* The output layer is fully connected to the hidden layer, and the hidden layer is fully connected to the input layer.
* The number of nodes in the input layer equals the number of features in the training data.
* The number of nodes in the hidden layer is set by the user. The default value is 100.
* The number of nodes equals the number of classes. For a two-class neural network, this means that all inputs must map to one of two nodes in the output layer.
* ***Custom definition script:*** Choose this option to create a custom neural network architecture, using the Net# language. With this option, you can define the number of hidden layers, their connections, and the mappings between layers.After selecting the custom script option, in the Neural network definition text box, type or paste Net# statements that define the network.

1. If you are not using the script option, use Number of hidden nodes, and type the number of hidden nodes. The default is one hidden layer with 100 nodes.
2. For Learning rate, define the size of the step taken at each iteration, before correction. A larger value for learning rate can cause the model to converge faster, but it can overshoot local minima.
3. For Number of learning iterations, specify the maximum number of times the algorithm should process the training cases.
4. For The initial learning weights diameter, specify the node weights at the start of the learning process.
5. For The momentum, specify a weight to apply during learning to nodes from previous iteration.
6. In The type of normalizer, select a method to use for feature normalization. The following normalization methods are supported:

* ***Binning normalizer:*** The binning normalizer creates bins of equal size, and then normalizes every value in each bin, dividing by the total number of bins.
* ***Gaussian normalizer:*** The Gaussian normalizer rescales the values of each feature to have mean 0 and variance 1. This is done by computing the mean and the variance of each feature. For each instance, the mean value is subtracted, and the result divided by the square root of the variance (the standard deviation).
* ***Min-max normalizer:*** The min-max normalizer linearly rescales every feature to the [0,1] interval. Rescaling to the [0,1] interval is done by shifting the values of each feature so that the minimal value is 0, and then dividing by the new maximal value (which is the difference between the original maximal and minimal values).
* ***Do not normalize:*** No normalization is performed.

1. Select the Shuffle examples option to shuffle cases between iterations. If you deselect this option, cases are processed in exactly the same order each time you run the experiment.
2. For Random number seed, type a value to use as the seed.

Specifying a seed value is useful when you want to ensure repeatability across runs of the same experiment. Otherwise, a system clock value is used as the seed, which can cause slightly different results each time you run the experiment.

1. Select the Allow unknown categorical levels option to create a grouping for unknown values in the training and validation sets. The model might

be less precise on known values but provide better predictions for new (unknown) values.

If you deselect this option, the model can accept only the values contained in the training data.

1. Add a tagged dataset to the experiment, and connect one of the training modules.

* If you set Create trainer mode to Single Parameter, use the Train Model module.
* If you set Create trainer mode to Parameter Range, use the Tune Model Hyperparameters module.

1. Run the experiment.

**Results**

After training is complete:

* To see a summary of the model's parameters, together with the feature weights learned from training, and other parameters of the neural network, right-click the output of Train Model or Tune Model Hyperparameters, and select Visualize.
* To save a snapshot of the trained model, right-click the Trained model output and select Save As Trained Model. This model is not updated on successive runs of the same experiment.
* To perform cross-validation against a labeled data set, connect the untrained model to Cross-Validate Model.

**Two-Class Support Vector Machine**

To create a model that is based on the support vector machine algorithm.

Support vector machines (SVMs) are a well-researched class of supervised learning methods. This particular implementation is suited to prediction of two possible outcomes, based on either continuous or categorical variables. After defining the model parameters, train the model by using one of the training modules, and providing a tagged dataset that includes a label or outcome column.

**Support vector machines**

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. This SVM model is a supervised learning model that requires labeled data. In the training process, the algorithm analyzes input data and recognizes patterns in a multi-dimensional feature space called the hyperplane. All input examples are represented as points in this space, and are mapped to output categories in such a way that categories are divided by as wide and clear a gap as possible. For prediction, the SVM algorithm assigns new examples into one category or the other, mapping them into that same space.

**How to configure Two-Class Support Vector Machine**

For this model type, it is recommended that you normalize the dataset before using it to train the classifier.

1. Add the Two-Class Support Vector Machine module to your experiment in Studio (classic).
2. Specify how you want the model to be trained, by setting the Create trainer mode option.

* Single Parameter: If you know how you want to configure the model, you can provide a specific set of values as arguments.
* Parameter Range: If you are not sure of the best parameters, you can find the optimal parameters by specifying multiple values and using the Tune Model Hyperparameters module to find the optimal configuration. The trainer iterates over multiple combinations of the settings and determines the combination of values that produces the best model.

1. For Number of iterations, type a number that denotes the number of iterations used when building the model. This parameter can be used to control trade-off between training speed and accuracy.
2. For Lambda, type a value to use as the weight for L1 regularization. This regularization coefficient can be used to tune the model. Larger values penalize more complex models.
3. Select the option, Normalize features, if you want to normalize features before training. If you apply normalization, before training, data points are centered at the mean and scaled to have one unit of standard deviation.
4. Select the option, Project to the unit sphere, to normalize coefficients.
5. Projecting values to unit space means that before training, data points are centered at 0 and scaled to have one unit of standard deviation.
6. In Random number seed, type an integer value to use as a seed if you want to ensure reproducibility across runs. Otherwise, a system clock value is used as a seed, which can result in slightly different results across runs.
7. Select the option, Allow unknown category, to create a group for unknown values in the training or validation sets. In this case, the model might be less precise for known values, but it can provide better predictions for new (unknown) values. If you deselect it, the model can accept only the values that are contained in the training data.
8. Select the option, Allow unknown category, to create a group for unknown values in the training or validation sets. In this case, the model might be less precise for known values, but it can provide better predictions for new (unknown) values.If you deselect it, the model can accept only the values that are contained in the training data.
9. Connect a labeled dataset, and one of the training modules:

* If you set Create trainer mode to Single Parameter, use the Train Model module.
* If you set Create trainer mode to Parameter Range, use the Tune Model Hyperparameters.

1. Run the experiment.

**Results**

After training is complete:

* To see a summary of the model's parameters, together with the feature weights learned from training, , right-click the output of Train Model or Tune Model Hyperparameters, and select Visualize.
* To use the trained models to make predictions, connect the trained model to the Score Model module.
* To perform cross-validation against a labeled data set, connect the untrained model and the dataset to Cross-Validate Model.