Creatio Academy

Machine learning

Machine learning service

Version 8.0

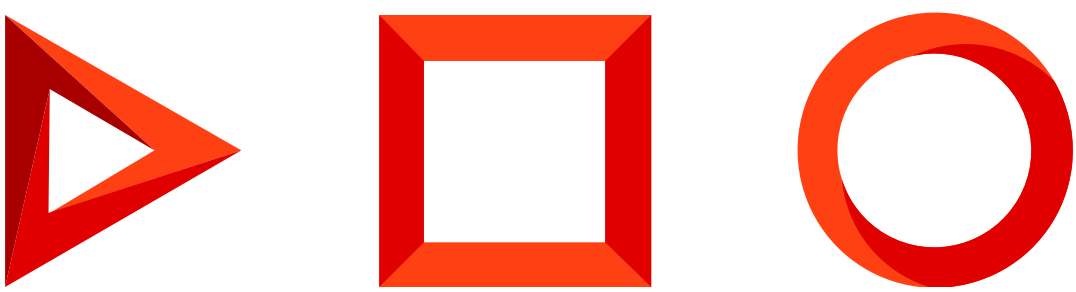


Table of Contents

Machine learning service

Advanced

Machine learning service basics

The machine learning (lookup value prediction) service uses statistical analysis methods for machine learning based on historical data. For example, a history of customer communications with customer support is considered historical data in Creatio. The message text, the date and the account category are used. The result is the [*Responsible Group*] field.

Creatio interaction with the prediction service

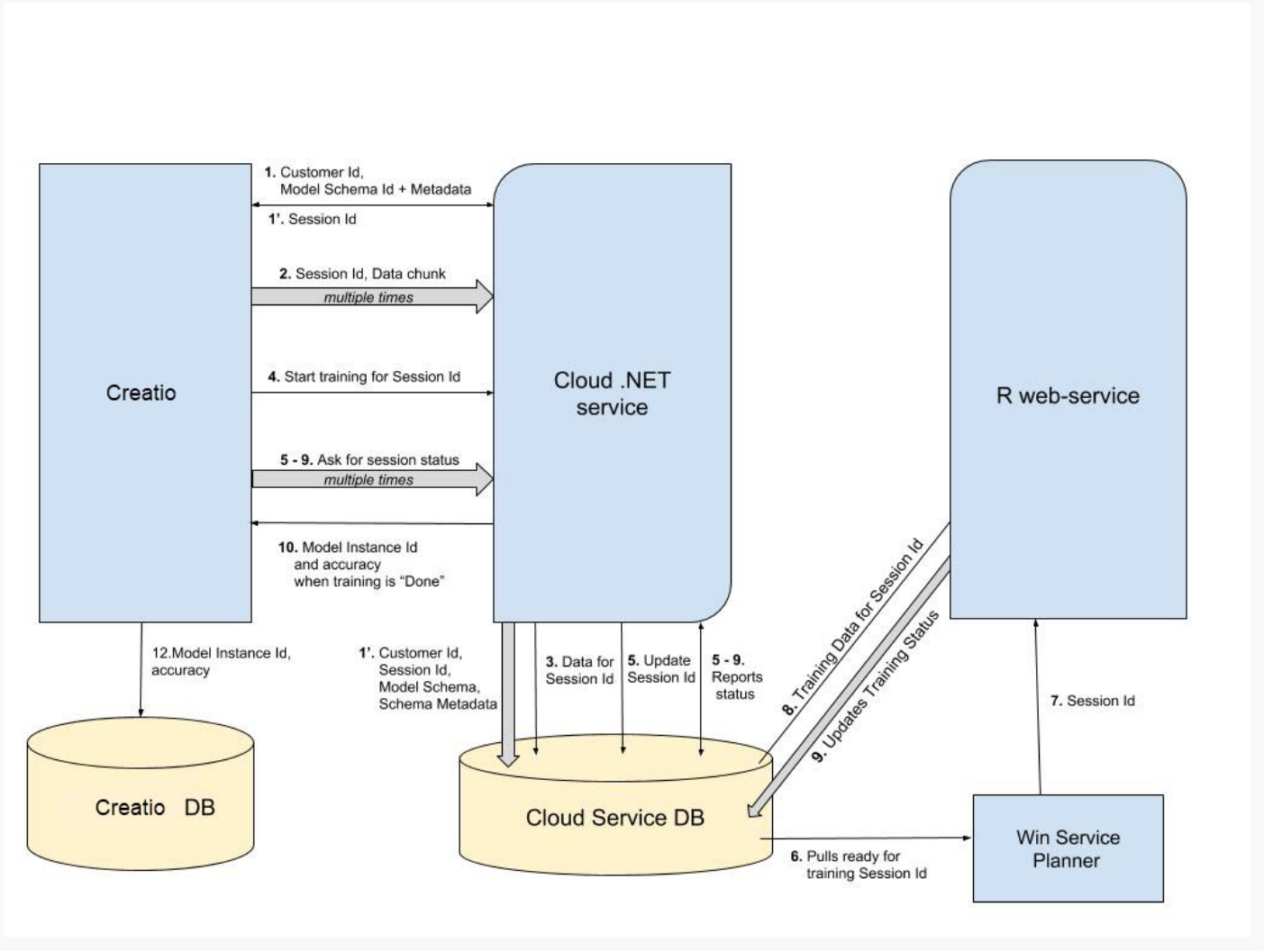
There are two stages of model processing in Creatio: training and prediction.

Prediction model is the algorithm which builds predictions and enables the system to automatically make decisions based on historical data.

Training

The service is “trained” at this stage. Main training steps:

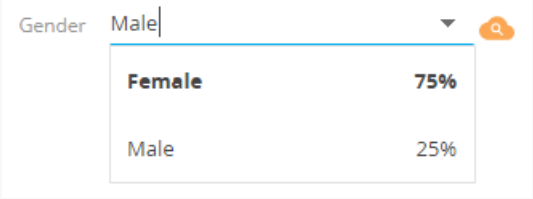
- Establishing a session for data transfer and training.
- Sequentially selecting a portion of data for the model and uploading it to the service.
- Requesting to include a model a training queue.
- `Training engine` processes the queue for model training, trains the model and saves its parameters to the local database.
- Creatio occasionally queries the service to get the model status.
- Once the model status is set to `Done`, the model is ready for prediction.



Prediction

The prediction task is performed through a call to the cloud service, indicating the Id of the model instance and the data for the prediction. The result of the service operation is a set of values with prediction probabilities, which is stored in Creatio in the `MLPrediction` table.

If there is a prediction in the `MLPrediction` table for a particular entity record, the predicted values for the field are automatically displayed on the edit page.



Creatio settings and data types for working with the prediction service

Creatio setup

The following data is provided for working with the prediction service in Creatio.

1. The `CloudServicesAPIKey` system setting authenticates the Creatio instance in cloud services.
2. The record in the [*ML problem types*] (`MLProblemType`) lookup with the populated [*ServiceUrl*] field is the address of the implemented prediction service.
3. The model records in the [*ML model*] (`MLModel`) lookup that contains information about the selected data for the model, the training period, the current training status, etc. For each model, the `MLProblemType` field must contain a reference to the correct record of the [*ML problem types*] lookup.
4. The `MLModelTrainingPeriodMinutes` system setting determines the frequency of model synchronization launch.

Expanding the training model logic

The above chain of classes calls and creates instances of each other through the IOC of the `Terrasoft.Core.Factories.ClassFactory` container.

If you need to replace the logic of any component, you need to implement the appropriate interface. When you start the application, you must bind the interface in your own implementation.

Interfaces for logic expansion:

`IMLModelTrainerJob` – the implementation of this interface will enable you to change the set of models for training.

`IMLModelTrainer` – responsible for the logic of loading data for training and updating the status of models.

`IMLServiceProxy` - the implementation of this interface will enable you to execute queries to arbitrary predictive services.

Auxiliary classes for forecasting

Auxiliary (utility) classes for forecasting enable you to implement two basic cases:

1. Prediction at the time of creating or updating an entity record on the server.
2. Prediction when the entity is changed on the edit page.

While predicting on the Creatio server-side, a business process is created that responds to the entity creation/change signal, reads a set of fields, and calls the prediction service. If you get the correct result, it stores the set of field values with probabilities in the `MLClassificationResult` table. If necessary, the business process records a separate value (for example, with the highest probability) in the corresponding field of the entity.

Creating data queries for the machine learning model

Use the Terrasoft.Core.DB.Select class instance for queries of training data or data for predicting machine learning service (see “Machine learning service”). It is dynamically imported by the `Terrasoft.Configuration.ML.QueryInterpreter`.

Attention. The `QueryInterpreter` interpreter does not allow the use of lambda expressions.

Use the provided `userConnection` variable as an argument of the `Terrasoft.Core.UserConnection` type in the `Select` constructor when building query expression. The column with the “Id” alias (the unique id of the target object instance) is required in the query expression.

The `Select` expression can be complex. Use the following practices to simplify it:

- Dynamic adding of types for the interpreter.
- Using local variables.
- Using the `Terrasoft.Configuration.QueryExtensions` utility class.

Dynamic adding of types for the interpreter

You can dynamically add types for the interpreter. For this, the `QueryInterpreter` class provides the `RegisterConfigurationType` and `RegisterType` methods. You can use them directly in the expression. For example, instead of direct using the type id:

```
new Select(userConnection)
        .Column("Id")
        .Column("Body")
    .From("Activity")
    .Where("TypeId").IsEqual(Column.Parameter("E2831DEC-CFC0-DF11-B00F-001D60E938C6"));
```

you can use the name of a constant from dynamically registered enumeration:

```
RegisterConfigurationType("ActivityConsts");
new Select(userConnection)
        .Column("Id")
        .Column("Body")
    .From("Activity")
    .Where("TypeId").IsEqual(Column.Parameter(ActivityConsts.EmailTypeUId));
```

Using local variables

You can use local variables to avoid code duplication and more convenient structuring. Constraint: the type of the variable must be statically calculated and defined by the `var` word.

For example, the query with repetitive use of delegates:

```
new Select(userConnection)
        .Column("Id")
        .Column("Body")
    .From("Activity")
    .Where("CreatedOn").IsGreater(Func.DateAddMonth(-1, Func.CurrentDateTime()))
    .And("StartDate").IsGreater(Func.DateAddMonth(-1, Func.CurrentDateTime()));
```

you can write in a following way:

```
var monthAgo = Func.DateAddMonth(-1, Func.CurrentDateTime());

new Select(userConnection)
        .Column("Id")
        .Column("Body")
    .From("Activity")
    .Where("StartDate").IsGreater(monthAgo)
    .And("ModifiedOn").IsGreater(monthAgo);
```

Connecting a custom web-service to the machine learning functionality

Attention. Creatio 7.16.2 and up supports connecting custom web-services to the machine learning functionality.

You can implement typical machine learning problems (classification, scoring, numerical regression) or other similar problems (for example, customer churn forecast) using a custom web-service. This article covers the procedure for connecting a custom web-service implementation of a prediction model to Creatio.

The main principles of the machine learning service operation are covered in the "Machine learning service" article.

The general procedure of connecting a custom web-service to the machine learning service is as follows:

1. Create a machine learning web-service engine.
2. Expand the problem type list of the machine learning service.
3. Implement a machine learning model.

Create a machine learning web-service engine

A custom web-service must implement a service contract for model training and making forecasts based on an existing prediction model. You can find a sample Swagger service contract of the Creatio machine learning service at https://demo-ml.bpmonline.com/swagger/index.html#/MLService

Required methods:

- `/session/start` – starting a model training session.
- `/data/upload` – uploading data for an active training session.
- `/session/info/get` – getting the status of the training session.
- `<training start custom method>` – a user-defined method to be called by Creatio when the data upload is over. The model training process must not be terminated until the execution of the method is complete. A training session may last for an indefinite period (minutes or even hours). When the training is over, the

`/session/info/get` method will return the training session status: either `Done` or `Error` depending on the result. Additionally, if the model is trained successfully, the method will return a model instance summary (`ModelSummary`): metrics type, metrics value, instance ID, and other data.

- `<Prediction custom method>` – an arbitrary signature method that will make predictions based on a trained prediction model referenced by the ID.

Web-service development with the Microsoft Visual Studio IDE is covered in the "Developing the configuration server code in the user solution" article.

Expanding the problem type list of the machine learning service

To expand the problem type list of the Creatio machine learning service, add a new record to the `MLProblemType` lookup. You must specify the following parameters:

- `Service endpoint Url` – the URL endpoint for the online machine learning service.
- `Training endpoint` – the endpoint for the training start method.
- `Prediction endpoint` – the endpoint for the prediction method.

Attention. You will need the ID of the new problem type record to proceed. Look up the ID in the [*dbo.MLProblemType*] DB table.

Implementing a machine learning model

To configure and display a machine learning model, you may need to extend the `MLModelPage` mini-page schema.

Implementing `IMLPredictor`

Implement the `Predict` method. The method accepts data exported from the system by object (formatted as `Dictionary<string, object>`, where `key` is the field name and `value` is the field value), and returns the prediction value. This method may use a proxy class that implements the `IMLServiceProxy` interface to facilitate web-service calls.

Implementing `IMLEntityPredictor`

Initialize the `ProblemTypeId` property with the ID of the new problem type record created in the `MLProblemType` lookup. Additionally, implement the following methods:

- `SaveEntityPredictedValues` – the method retrieves the prediction value and saves it for the system `entity`, for which the prediction process is run. If the returned value is of the `double` type or is similar to classification results, you can use the methods provided in the `PredictionSaver` auxiliary class.
- `SavePrediction` **optional** – the method saves the prediction value with a reference to the trained model instance and the ID of the entity (`entityId`). For basic problems, the system provides the `MLPrediction` and `MLClassificationResult` entities.

Extending`IMLServiceProxy` and `MLServiceProxy` optional

You can extend the existing `IMLServiceProxy` interface and the corresponding implementations in the prediction method of the current problem type. In particular, the `MLServiceProxy` class provides the `Predict` generic method that accepts contracts for input data and for prediction results.

Implementing `IMLBatchPredictor`

If the web-service is called with a large set of data (500 instances and more), implement the `IMLBatchPredictor` interface. You must implement the following methods:

- `FormatValueForSaving` – returns a converted prediction value ready for database storage. In case a batch prediction process is running, the record is updated using the `Update` method rather than `Entity` instances to speed up the process.
- `SavePredictionResult` – defines how the system will store the prediction value per entity. For basic ML problems, the system provides `MLPrediction` and `MLClassificationResult` objects.

Examples of data queries for the machine learning model

Advanced

Using the `Terrasoft.Configuration.QueryExtensions` utility class

The `Terrasoft.Configuration.QueryExtensions` utility class provides several extending methods for the `Terrasoft.Core.DB.Select`. This enables to build more compact queries.

As the `object sourceColumn` argument you can use following types (they will be transformed to the `Terrasoft.Core.DB.QueryColumnExpression`) for all extending methods:

- `System.String` – the name of the column in the `TableAlias.ColumnName as ColumnAlias` format (where the `TableAlias` and `ColumnAlias` are optional) or "*" – all columns.
- `Terrasoft.Core.DB.QueryColumnExpression` – will be added without changes.
- `Terrasoft.Core.DB.IQueryColumnExpressionConvertible` – will be converted.
- `Terrasoft.Core.DB.Select` – will be considered as subquery.

Attention. An exception will be thrown if the type is not supported.

static Select Cols(this Select select, params object[] sourceColumns)

Adds specified columns or subexpressions to the query.

Using the `Cols()` extension method, instead of the following expression:

```
new Select(userConnection)
        .Column("L", "Id")
        .Column("L", "QualifyStatusId")
        .Column("L", "LeadTypeId")
        .Column("L", "LeadSourceId")
        .Column("L", "LeadMediumId").As("LeadChannel")
        .Column("L", "BusinesPhone").As("KnownBusinessPhone")
    .From("Lead").As("L");
```

you can write:

```
new Select(userConnection).Cols(
        "L.Id",
        "L.QualifyStatusId",
        "L.LeadTypeId",
        "L.LeadSourceId",
        "L.LeadMediumId AS LeadChannel",
        "L.BusinesPhone AS KnownBusinessPhone")
    .From("Lead").As("L");
```

static Select Count(this Select select, object sourceColumn)

Adds an aggregation column to calculate the number of non-empty values to the query.

For example, instead:

```
var activitiesCount = new Select(userConnection)
    .Column(Func.Count(Column.Asterisk()))
    .From("Activity")
```

you can write:

```
var activitiesCount = new Select(userConnection)
    .Count("*") // You can also specify the column name.
    .From("Activity")
```

static Select Coalesce(this Select select, params object[] sourceColumns)

Adds a column with the function of determining the first value not equal to NULL to the query.

For example, instead:

```
new Select(userConnection)
    .Cols("L.Id")
    .Column(Func.Coalesce(
            Column.SourceColumn("L", "CountryId"),
            Column.SourceColumn("L", "CountryId"),
            Column.SourceColumn("L", "CountryId")))
        .As("CountryId")
    .From("Lead").As("L")
        .LeftOuterJoin("Contact").As("C").On("L", "QualifiedContactId").IsEqual("C", "Id")
         .LeftOuterJoin("Account").As("A").On("L", "QualifiedAccountId").IsEqual("A", "Id");
```

you can write:

```
new Select(userConnection)
        .Cols("L.Id")
        .Coalesce("L.CountryId", "C.CountryId", "A.CountryId").As("CountryId")
    .From("Lead").As("L")
        .LeftOuterJoin("Contact").As("C").On("L", "QualifiedContactId").IsEqual("C", "Id")
        .LeftOuterJoin("Account").As("A").On("L", "QualifiedAccountId").IsEqual("A", "Id");
```

static Select DateDiff(this Select select, DateDiffQueryFunctionInterval interval, object startDateExpression, object endDateExpression)

Adds a column that specifies the date difference to the query.

For example, instead:

```
new Select(_userConnection)
        .Cols("Id")
        .Column(Func.DateDiff(DateDiffQueryFunctionInterval.Day,
            Column.SourceColumn("L", "CreatedOn"), Func.CurrentDateTime())).As("LeadAge")
    .From("Lead").As("L");
```

you can write:

```
var day = DateDiffQueryFunctionInterval.Day;
new Select(userConnection)
        .Cols("L.Id")
        .DateDiff(day, "L.CreatedOn", Func.CurrentDateTime()).As("LeadAge")
    .From("Lead").As("L");
```

static Select IsNull(this Select select, object checkExpression, object replacementValue)

Adds a column with the function replacing NULL value with a replacement expression.

For example, instead:

```
new Select(userConnection).Cols("Id")
        .Column(Func.IsNull(
               Column.SourceColumn("L", "CreatedOn"),
               Column.SourceColumn("L", "ModifiedOn")))
    .From("Lead").As("L");
```

you can write:

```
new Select(userConnection).Cols("L.Id")
        .IsNull("L.CreatedOn", "L.ModifiedOn")
    .From("Lead").As("L");
```

Connect a web service to the machine learning functionality

Medium

Case. Connect a custom implementation of predictive scoring to Creatio.

Source code

You can download the package with an implementation of the case using the following link.

Case implementation algorithm

1. Create a machine learning web service engine

A sample implementation of a machine learning web service for ASP.Net Core 3.1 can be downloaded here.

Implement the `MLService` machine learning web service.

Declare the required methods:

- `/session/start`
- `/data/upload`

- `/session/info/get`
- `/fakeScorer/beginTraining`
- `/fakeScorer/predict`

The complete source code of the module is available below:

```
namespace FakeScoring.Controllers
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Net;
    using Microsoft.AspNetCore.Mvc;
    using Terrasoft.ML.Interfaces;
    using Terrasoft.ML.Interfaces.Requests;
    using Terrasoft.ML.Interfaces.Responses;

    [ApiController]
    [Route("")]
    public class MLService : Controller
    {
        public const string FakeModelInstanceUId = "{BFC0BD71-19B1-47B1-8BC4-D761D9172667}";

        private List<ScoringOutput.FeatureContribution> GenerateFakeContributions(DatasetValue r
            var random = new Random(42);
            return record.Select(columnValue => new ScoringOutput.FeatureContribution {
                Name = columnValue.Key,
                Contribution = random.NextDouble(),
                Value = columnValue.Value.ToString()
            }).ToList();
        }

        [HttpGet("ping")]
        public JsonResult Ping() {
            return new JsonResult("Ok");
        }

        /// <summary>
        /// Handshake request to service with the purpose to start a model training session.
        /// </summary>
        /// <param name="request">Instance of <see cref="StartSessionRequest"/>.</param>
        /// <returns>Instance of <see cref="StartSessionResponse"/>.</returns>
        [HttpPost("session/start")]
        public StartSessionResponse StartSession(StartSessionRequest request) {
            return new StartSessionResponse {
                SessionId = Guid.NewGuid()
            };
```

```
        }

        /// <summary>
        /// Uploads training data.
        /// </summary>
        /// <param name="request">The upload data request.</param>
        /// <returns>Instance of <see cref="JsonResult"/>.</returns>
        [HttpPost("data/upload")]
        public JsonResult UploadData(UploadDataRequest request) {
            return new JsonResult(string.Empty) {
                StatusCode = (int)HttpStatusCode.OK
            };
        }

        /// <summary>
        /// Begins fake scorer training on uploaded data.
        /// </summary>
        /// <param name="request">The scorer training request.</param>
        /// <returns>Simple <see cref="JsonResult"/>.</returns>
        [HttpPost("fakeScorer/beginTraining")]
        public JsonResult BeginScorerTraining(BeginScorerTrainingRequest request) {
            // Start training work here. It doesn't have to be done by the end of this request.
            return new JsonResult(string.Empty) {
                StatusCode = (int)HttpStatusCode.OK
            };
        }

        /// <summary>
        /// Returns current session state and model statistics, if training is complete.
        /// </summary>
        /// <param name="request">Instance of <see cref="GetSessionInfoRequest"/>.</param>
        /// <returns>Instance of <see cref="GetSessionInfoResponse"/> with detailed state info.<
        [HttpPost("session/info/get")]
        public GetSessionInfoResponse GetSessionInfo(GetSessionInfoRequest request) {
            var response = new GetSessionInfoResponse {
                SessionState = TrainSessionState.Done,
                ModelSummary = new ModelSummary {
                    DataSetSize = 100500,
                    Metric = 0.79,
                    MetricType = "Accuracy",
                    TrainingTimeMinutes = 5,
                    ModelInstanceUId = new Guid(FakeModelInstanceUId)
                }
            };
            return response;
        }

        /// <summary>
        /// Performs fake scoring prediction.
```

```
        /// </summary>
        /// <param name="request">Request object.</param>
        /// <returns>Scoring rates.</returns>
        [HttpPost("fakeScorer/predict")]
        public ScoringResponse Predict(ExplainedScoringRequest request) {
            List<ScoringOutput> outputs = new List<ScoringOutput>();
            var random = new Random(42);
            foreach (var record in request.PredictionParams.Data) {
                var output = new ScoringOutput {
                    Score = random.NextDouble()
                };
                if (request.PredictionParams.PredictContributions) {
                    output.Bias = 0.03;
                    output.Contributions = GenerateFakeContributions(record);
                }
                outputs.Add(output);
            }
            return new ScoringResponse { Outputs = outputs };
        }
    }
}
```

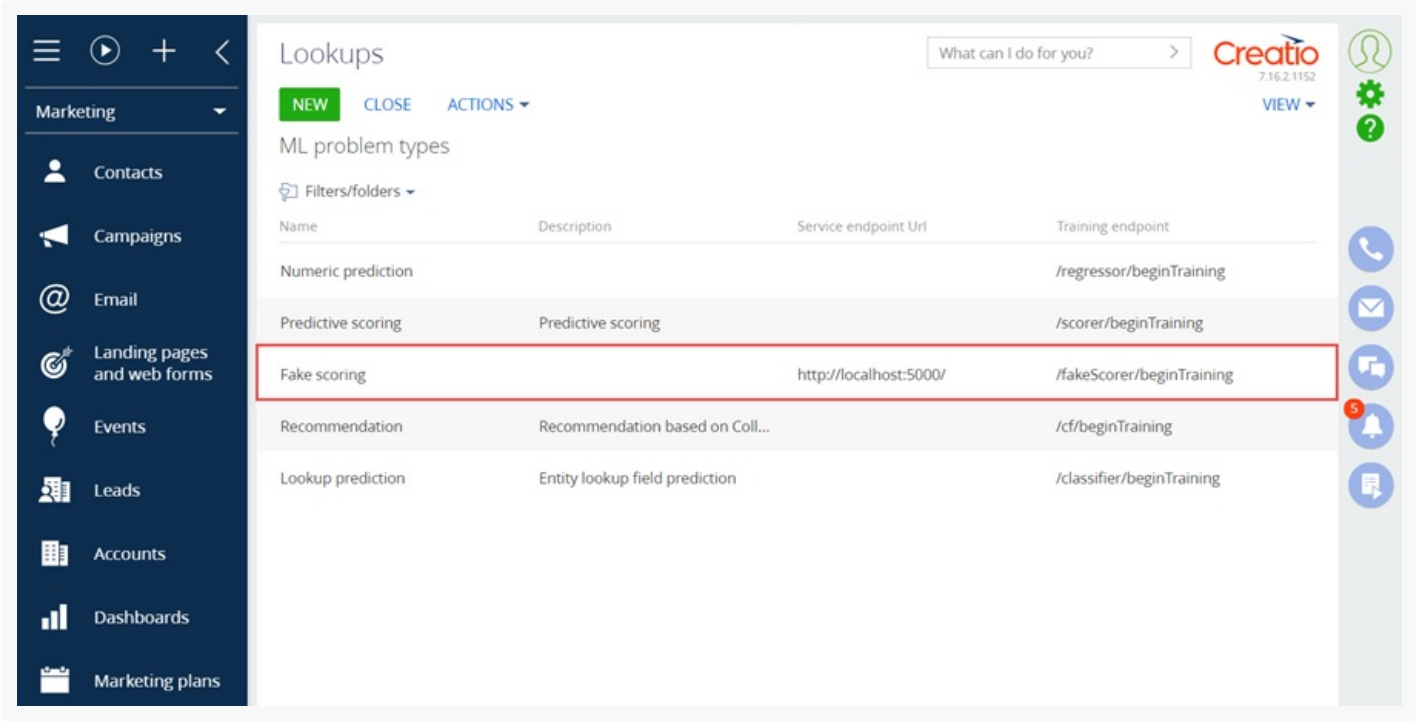
2. Expand the problem list of the machine learning service.

To expand the problem list:

1. Open the System Designer by clicking . Go to the [*System setup*] block –> click [*Lookups*].
2. Select the [*ML problem types*] lookup.
3. Add a new record.

In the record, specify:

- [*Name*] – "Fake scoring".
- [*Service endpoint Url*] – "http://localhost:5000/".
- [*Training endpoint*] – "/fakeScorer/beginTraining".



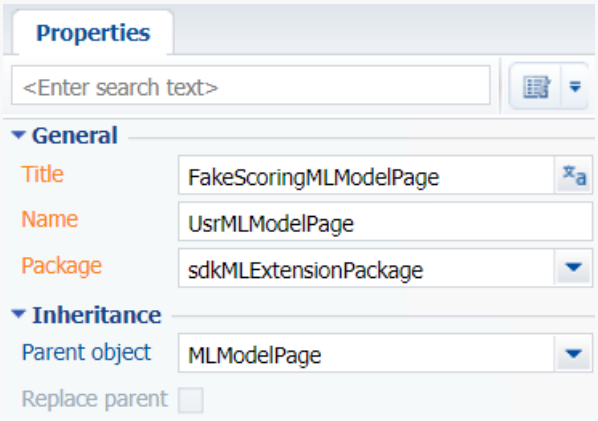
The ID of the added record is `319c39fd-17a6-453a-bceb-57a398d52636`.

3. Implement a machine learning model

Execute the [*Add*] –> [*Additional*] –> [*Schema of Edit Page View Model*] menu command on the [*Schemas*] tab in the [*Configuration*] section of the custom package. The newly created module should inherit the `MLModelPage` base page functionality defined in the `ML` package. Specify this schema as the parent one for a new schema.

Specify the following parameters for the created object schema:

- [*Title*] – “FakeScoringMLModelPage”.
- [*Name*] – "UsrMLModelPage".
- [*Parent object*] – select “MLModelPage”.



Overload the `getIsScoring` base method to make the style of the new mini-page identical to the mini-page for creating a predictive scoring model. The complete source code of the module is available below:

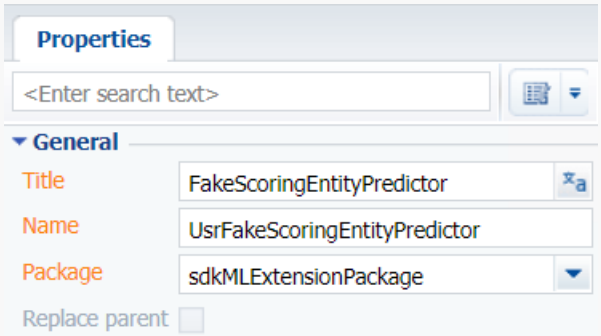
```
Type your example code here. It will be automatically colorized when you switch to Preview or bu
```

After making changes, save and publish the schema.

Go to the [*Advanced settings*] section -> [*Configuration*] -> Custom package -> the [*Schemas*] tab. Click [*Add*] —> [*Source Code*]. Learn more about creating a schema of the ([*Source Code*] type in the "Create the ([*Source Code*] schema" article.

Specify the following parameters for the created object schema:

- [*Title*] – "FakeScoringEntityPredictor".
- [*Name*] – "UsrFakeScoringEntityPredictor".



Implement the `Predict` method that accepts data exported from the system by object and returns the prediction value. The method uses a proxy class that implements the `IMLServiceProxy` interface to facilitate web service calls.

Initialize the `ProblemTypeId` property with the ID of the new problem type record created in the `MLProblemType` lookup and implement the `SaveEntityPredictedValues` and `SavePrediction` methods. The complete source code of the module is available below:

```
namespace Terrasoft.Configuration.ML
{
    using System;
    using System.Collections.Generic;
    using Core;
    using Core.Factories;
    using Terrasoft.ML.Interfaces;

    [DefaultBinding(typeof(IMLEntityPredictor), Name = "319C39FD-17A6-453A-BCEB-57A398D52636")]
    [DefaultBinding(typeof(IMLPredictor<ScoringOutput>), Name = "319C39FD-17A6-453A-BCEB-57A398D
    public class FakeScoringEntityPredictor: MLBaseEntityPredictor<ScoringOutput>, IMLEntityPred
        IMLPredictor<ScoringOutput>
    {
```

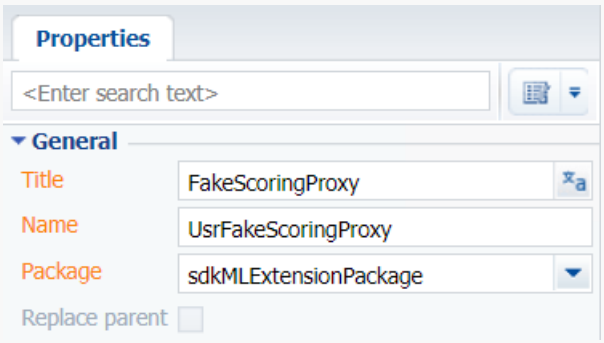
```
        public FakeScoringEntityPredictor(UserConnection userConnection) : base(userConnection)
        }

        protected override Guid ProblemTypeId => new Guid("319C39FD-17A6-453A-BCEB-57A398D52636"

        protected override ScoringOutput Predict(IMLServiceProxy proxy, MLModelConfig model,
                Dictionary<string, object> data) {
            return proxy.FakeScore(model, data, true);
        }

        protected override List<ScoringOutput> Predict(MLModelConfig model,
                IList<Dictionary<string, object>> dataList, IMLServiceProxy proxy) {
            return proxy.FakeScore(model, dataList, true);
        }

        protected override void SaveEntityPredictedValues(MLModelConfig model, Guid entityId,
                ScoringOutput predictedResult) {
            var predictedValues = new Dictionary<MLModelConfig, double> {
                { model, predictedResult.Score }
            };
            PredictionSaver.SaveEntityScoredValues(model.EntitySchemaId, entityId, predictedValu
        }

        protected override void SavePrediction(MLModelConfig model, Guid entityId, ScoringOutput
            PredictionSaver.SavePrediction(model.Id, model.ModelInstanceUId, entityId, predicted
        }

    }
}
```

After making changes, save and publish the schema.

Go to the [*Advanced settings*] section -> [*Configuration*] -> Custom package -> the [*Schemas*] tab. Click [*Add*] -> [*Source code*]. Learn more about creating a schema of the [*Source Code*] type in the "Create the [*Source code*] schema" article.

Specify the following parameters for the created object schema:

- [*Title*] – "FakeScoringProxy".
- [*Name*] – "UsrFakeScoringProxy".



Extend the existing `IMLServiceProxy` interface and the corresponding implementations in the prediction method of the current problem type. In particular, the `MLServiceProxy` class provides the `Predict` generic method that accepts contracts for input data and prediction results.

The complete source code of the module is available below:

```
namespace Terrasoft.Configuration.ML
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using Terrasoft.ML.Interfaces;
    using Terrasoft.ML.Interfaces.Requests;
    using Terrasoft.ML.Interfaces.Responses;

    public partial interface IMLServiceProxy
    {
        ScoringOutput FakeScore(MLModelConfig model, Dictionary<string, object> data, bool predi
        List<ScoringOutput> FakeScore(MLModelConfig model, IList<Dictionary<string, object>> dat
            bool predictContributions);
    }

    public partial class MLServiceProxy : IMLServiceProxy
    {
        public ScoringOutput FakeScore(MLModelConfig model, Dictionary<string, object> data, boo
            ScoringResponse response = Predict<ExplainedScoringRequest, ScoringResponse>(model.M
                new List<Dictionary<string, object>> { data }, model.PredictionEndpoint,
                100, predictContributions);
            return response.Outputs.FirstOrDefault();
        }

        public List<ScoringOutput> FakeScore(MLModelConfig model, IList<Dictionary<string, objec
                bool predictContributions) {
            int count = Math.Min(1, dataList.Count);
            int timeout = Math.Max(ScoreTimeoutSec * count, BatchScoreTimeoutSec);
            ScoringResponse response = Predict<ScoringRequest, ScoringResponse>(model.ModelInsta
                dataList, model.PredictionEndpoint, timeout, predictContributions);
            return response.Outputs;
```

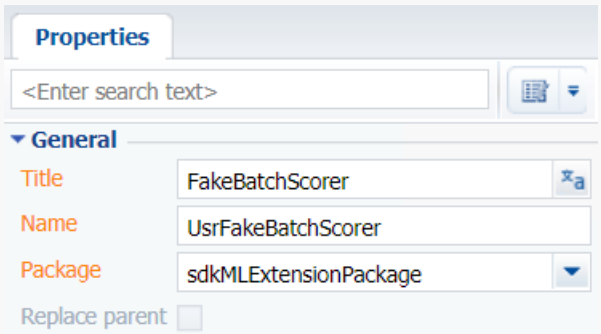
```
        }
    }
}
```

After making changes, save and publish the schema.

Go to the [*Advanced settings*] section -> [*Configuration*] -> Custom package -> the [*Schemas*] tab. Click [*Add*] -> [*Source code*]. Learn more about creating a schema of the [*Source Code*] type in the "Create the [*Source code*] schema" article.

Specify the following parameters for the created object schema:

- [*Title*] – "FakeBatchScorer".
- [*Name*] – "UsrFakeBatchScorer".



To use the batch prediction functionality, implement the `IMLBatchPredictor` interface, and the `FormatValueForSaving` and `SavePredictionResult` methods.

The complete source code of the module is available below:

```
namespace Terrasoft.Configuration.ML
{
    using System;
    using Core;
    using Terrasoft.Core.Factories;
    using Terrasoft.ML.Interfaces;

    [DefaultBinding(typeof(IMLBatchPredictor), Name = "319C39FD-17A6-453A-BCEB-57A398D52636")]
    public class FakeBatchScorer : MLBatchPredictor<ScoringOutput>
    {

        public FakeBatchScorer(UserConnection userConnection) : base(userConnection) {
        }

        protected override object FormatValueForSaving(ScoringOutput scoringOutput) {
            return Convert.ToInt32(scoringOutput.Score * 100);
        }
```

```
        protected override void SavePredictionResult(Guid modelId, Guid modelInstanceUId, Guid e
                ScoringOutput value) {
            PredictionSaver.SavePrediction(modelId, modelInstanceUId, entityId, value);
        }

    }

}
```

After making changes, save and publish the schema.

Implement custom prediction model

Advanced

Case description

Implement automatic prediction for the [*AccountCategory*] column by the values of the [*Country*], [*EmployeesNumber*] and [*Industry*] field while saving the account record. The following conditions should be met:

- Model learning should be created on the base of account records for last 90 days.
- Moodel Retraining should be performed every 30 days.
- Permissible value of prediction accuracy for the model – 0,6.

Attention.

To complete this case you need to check the correctness of the value of the [*Creatio cloud services API key*] (`CloudServicesAPIKey` code) system setting and the URL of the predictive service in the [*Service endpoint Url*] field of the [*ML problem types*] lookup.

Note. You can use the page UI-tools (fields and filters), as well as the [*Predict data*] business process element to perform the described case. You can find more cases on implementing prediction using the default Creatio tools in the "Predictive analysis" article.

Case implementation algorithm

1. Model learning

To train the model:

1. Add a record to the [*ML Model*] lookup. Values of the record fields are given in the Table 1.

Table 1. Values of the record fields of the ML Model lookup

Table 1. Values of the record fields of the ML model lookup

Field	Value
Name	Predict account category
ML problem type	Lookup prediction
Target schema for prediction	Account
Quality metric low limit	0,6
Model retrain frequency (days)	30

Training set metadata	<pre>{
	"inputs": [
	{
	"name": "CountryId",
	"type": "Lookup",
	"isRequired": true
	},
	{
	"name": "EmployeesNumberId",
	"type": "Lookup",
	"isRequired": true
	},
	{
	"name": "IndustryId",
	"type": "Lookup",
	"isRequired": true
	}
	],
	"output": {
	"name": "AccountCategoryId",
	"type": "Lookup",
	"displayName": "AccountCategory"
	}
	}

Training set query	`new Select(userConnection)`
	`  .Column("a", "Id").As("Id")`
	`  .Column("a", "CountryId")`
	`  .Column("a", "EmployeesNumberId")`
	`  .Column("a", "IndustryId")`
	`  .Column("a", "AccountCategoryId")`
	`  .Column("c", "Name").As("AccountCategory")`
	`.From("Account").As("a")`
	`.InnerJoin("AccountCategory").As("c").On("c", "Id").IsEqual("a", "AccountCateg`
	`.Where("a", "CreatedOn").IsGreater(Column.Parameter(DateTime.Now.AddDays(-90))`
	
	You can find examples of queries in the Creating data queries for the machine learning model

Predictions enabled (checkbox)	Enable

2. Perform the [*Execute model training job*] action on the [`ML Model`] lookup field.

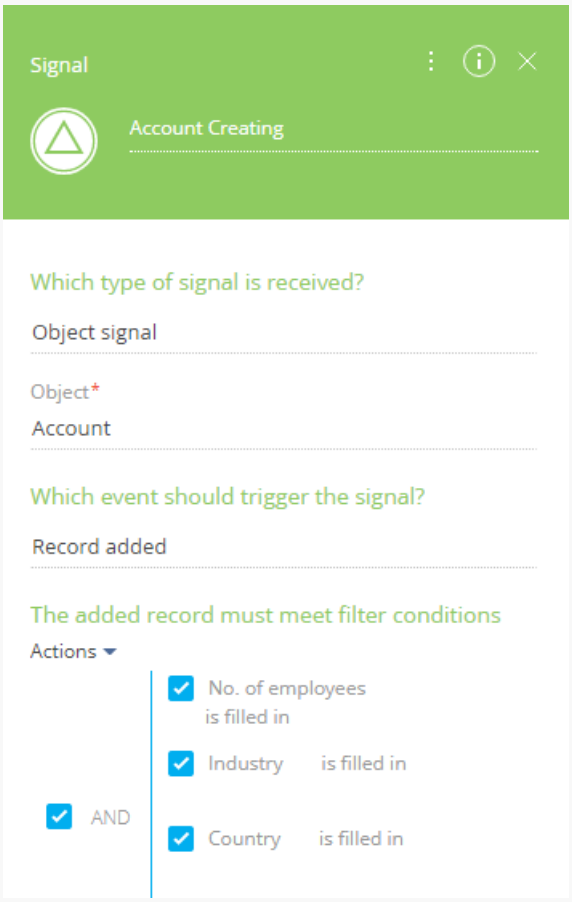
Wait until the values of the [*Model processing status*] field will be changed in following sequence: `DataTransfer`, `QueuedToTrain`, `Training`, `Done`. The process may take several hours to finish (it depends on the amount of passed data and general workload of the predictive service.

2. Performing the prediction

To start the predictions:

1. Create a business process in the user package. Select the saving of the [*Contact*] object as a start signal for the process. Check if the required fields are populated (Fig. 1).

Fig. 1. Start signal properties.



2. Add the `MLModelId` lookup parameter that refers to the [*ML Model*] entity. Select the record with the [*Predict account category*] model as a value.

3. Add the `RecordId` lookup parameter that refers to the [*Account*] entity. Select a reference for the `RecordId` parameter of the [*Signal*] element as a value.

4. Add a [*Script task*] element on the business process diagram and add the following code there:

```
var userConnection = Get<UserConnection>("UserConnection");
// Getting the Id of the Account record.
Guid entityId = Get<Guid>("RecordId");
// Geeting the id of the model.
var modelId = Get<Guid>("MLModelId");
var connectionArg = new ConstructorArgument("userConnection", userConnection);
// Object for calling prediction.
var predictor = ClassFactory.Get<MLEntityPredictor>(connectionArg);
// Call of the forecasting service. The Data is saved in MLPrediction and in case of high probab
predictor.PredictEntityValueAndSaveResult(modelId, entityId);
return true;
```

After saving and compiling the process, the prediction will be performed for new accounts. The prediction will be displayed on the account edit page.

Attention.

This implementation of the prediction slows down the saving an account record because call of the prediction service is executed in 2 seconds. This can reduce the performance of the mass operations with data saving, like import from Excel.

IMLModelTrainerJob class C#

Advanced

Implements the `IJobExecutor` and `IMLModelTrainerJob` interfaces.

It is used for model synchronization tasks.

Orchestrates model processing on the side of Creatio by launching data transfer sessions, starting trainings, and also checking the status of the models processed by the service. Instances are launched by default by the task scheduler through the standard `Execute` method of the `IJobExecutor` interface.

Methods

`IMLModelTrainerJob.RunTrainer()`

A virtual method that encapsulates the synchronization logic. The base implementation of this method performs the following actions:

1. Selecting models for training – the records are selected from `MLModel` based on the following filter:
   - The `MetaData` and `TrainingSetQuery` fields are populated.
   - The `Status` field is not in the `NotStarted`, `Done` or `Error` state (or not populated at all).
   - `TrainFrequency` is more than 0.
   - The `TriedToTrainOn` days have passed since the last training date (`TrainFrequency`).
     For each record of this selection, the data is sent to the service with the help of the predictive model trainer.
2. Selecting previously trained models and updating their status (if necessary).

The data transfer session for the selection starts for each suitable model. The data is sent in packages of 1000 records during the session. For each model, the selection size is limited to 75,000 records.

IMLModelTrainer class C#

Advanced

Implements the `IMLModelTrainer` interface.

Trainer of the prediction model.

Responsible for the overall processing of a single model during the training stage. Communication with the service is provided through a proxy to a predictive service.

Methods

`IMLModelTrainer.StartTrainSession()`

Sets the training session for the model.

`IMLModelTrainer.UploadData()`

Transfers the data according to the model selection in packages of 1000 records. The selection is limited to 75,000 records.

`IMLModelTrainer.BeginTraining()`

Indicates the completion of data transfer and informs the service about the need to put the model in the training queue.

`IMLModelTrainer.UpdateModelState()`

Requests the service for the current state of the model and updates the `Status` (if necessary).

If the training was successful (`Status` contains the `Done` value), the service returns the metadata for the trained instance, particularly the accuracy of the resulting instance. If the precision is greater than or equal to the lower threshold (`MetricThreshold`), the ID of the new instance is written in the `ModelInstanceUId` field.

IMLServiceProxy class C#

Advanced

Implements the `IMLServiceProxy` interface.

It is the proxy to the prediction service.

A wrapper class for http requests to a prediction service.

Methods

`IMLServiceProxy.UploadData()`

Sends a data package for the training session.

`IMLServiceProxy.BeginTraining()`

Calls the service for setting up training in the queue.

`IMLServiceProxy.GetTrainingSessionInfo()`

Requests the current state from the service for the training session.

`IMLServiceProxy.Classify(Guid modelInstanceUId, Dictionary<string, object> data)`

Calls the prediction service of the field value for a single set of field values for the previously trained model instance. In the `Dictionary data` parameter, the field name is passed as the key, which must match the name specified in the `MetaData` field of the model lookup. If the result is successful, the method returns a list of values with the `ClassificationResult` type.

The `ClassificationResult` properties

`Value`

Field value.

`Probability`

The probability of a given value in the [0:1] range. The sum of the probabilities for one list of results is close to 1 (values of about 0 can be omitted).

`Significance`

The level of importance of this prediction.

`Significance` enumeration

`High`	This field value has a distinct advantage over other values from the list. Only one element in the prediction list can have this level.
`Medium`	The value of the field is close to several other high values in the list. For example, two values in the list have a probability of 0.41 and 0.39, and all the others are significantly smaller.
`None`	Irrelevant values with low probabilities.

IMLEntityPredictor class C#

Advanced

A utility class that helps to predict the value of a field based on a particular model (either one or several models) for a particular entity.

Methods

`PredictEntityValueAndSaveResult(Guid modelId, Guid entityId)`

Based on the model `Id` and entity `Id`, performs predictions and records the results in the resulting entity field. Works with any machine learning task: classification, scoring, numeric field prediction.

`ClassifyEntityValues(List<Guid> modelIds, Guid entityId)`

Based on the model (or list of several models created for the same object) `Id` and entity `Id` performs classification and returns the glossary, whose key is the model object, and the values are the predicted values.

IMLPredictionSaver class C#

Advanced

The utility class that assists to save the prediction results in the Creatio object.

Methods

`SaveEntityPredictedValues(Guid schemaUId, Guid entityId, Dictionary<MLModelConfig, List<Classifi Func<Entity, string, ClassificationResult, bool> onSetEntityValue)`

Saves the `MLEntityPredictor.ClassifyEntityValues` classification results in the Creatio object. By default, it saves only the result, whose `Significance` equals to [*High*]. You can still override this behavior using the passed `onSetEntityValue` delegate. If the delegate returns `false`, the value will not be recorded in the Creatio object.

Properties of the ClassificationResult type

`Value`	Field value.
`Probability`	The probability of a given value in the [0:1] range. The sum of the probabilities for one list of results is close to 1 (values of about 0 can be omitted).
`Significance`	The level of importance of this prediction.

Significance enumeration

`High`	This field value has a distinct advantage over other values from the list. Only one element in the prediction list can have this level.
`Medium`	The value of the field is close to several other high values in the list. For example, two values in the list have a probability of 0.41 and 0.39, and all the others are significantly smaller.
`None`	Irrelevant values with low probabilities.

MLModel lookup C#

Advanced

MLModel lookup

Contains information about the selected data for the model, the training period, the current training status, etc.

Main MLModel lookup fields

`Name` `String`

Model name.

`ModelInstanceUId` `Unique identifier`

The identifier of the current model instance.

`TrainedOn` `Date/time`

The date/time of last training attempt.

`TriedToTrainOn` `Date/time`

The date/time of last training attempt.

`TrainFrequency` `Integer`

Model retraining frequency (days).

`MetaData` `String`

Metadata with selection column types.

Metadata JSON format

JSON

```
{
    inputs: [
        {
            name: "Name of the field 1 in the data sample",
            type: "Text",
            isRequired: true
        },
        {
            name: "Name of the field 2 in the data sample",
            type: "Lookup"
        },
        //...
    ],
    output: {
        name: "Resulting field",
        type: "Lookup",
        displayName: "Name of the column to display"
    }
}
```

In this code:

- `inputs` – a set of incoming columns for the model.
- `output` – a column, the value of which the model should predict.

Column descriptions support the following attributes:

- `name` – field name from the `TrainingSetQuery` expression.
- `type` – data type for the training engine.

Supported values

`Text`	text column;
`Lookup`	lookup column;
`Boolean`	logical data type;
`Numeric`	numeric type;
`DateTime`	date and time.

- `isRequired` – mandatory field value (`true`/`false`). Default value – `false`.

`TrainingSetQuery` `String`

C#-expression of the training data selection. This expression should return the `Terrasoft.Core.DB.Select` class instance.

TrainingSetQuery example

C#

```
(Select)new Select(userConnection)
.Column("Id")
.Column("Symptoms")
.Column("CreatedOn")
.From("Case", "c")
.OrderByDesc("c", "CreatedOn")
```

Attention. Select the “Unique identifier” column type in the selection expression. This column should have the `Id` name.

Attention. If the selection expression contains a column for sorting, then this column must be present in the resulting selection.

You can find examples of queries in the "**Examples of data queries for the machine learning model**" example.

`RootSchemaUId` `Unique identifier`

A link to an object schema for which the prediction will be executed.

`Status` `String`

The status of model processing (data transfer, training, ready for forecasting).

`InstanceMetric` `Number`

A quality metric for the current model instance.

`MetricThreshold` `Number`

Lowest threshold of model quality.

`PredictionEnabled` `Logical`

A flag that includes the prediction for this model.

`TrainSessionId` `Unique identifier`

Current training session.

`MLProblemType` `Unique identifier`

Machine learning problem (defines the algorithm and service url for model training).