

# Сервис машинного обучения

Версия 8.0



Эта документация предоставляется с ограничениями на использование и защищена законами об интеллектуальной собственности. За исключением случаев, прямо разрешенных в вашем лицензионном соглашении или разрешенных законом, вы не можете использовать, копировать, воспроизводить, переводить, транслировать, изменять, лицензировать, передавать, распространять, демонстрировать, выполнять, публиковать или отображать любую часть в любой форме или посредством любые значения. Обратный инжиниринг, дизассемблирование или декомпиляция этой документации, если это не требуется по закону для взаимодействия, запрещены.

Информация, содержащаяся в данном документе, может быть изменена без предварительного уведомления и не может гарантировать отсутствие ошибок. Если вы обнаружите какие-либо ошибки, сообщите нам о них в письменной форме.

# Содержание

<b>Сервис машинного обучения</b>	<b>4</b>
Общие принципы работы сервиса машинного обучения	4
Составление запросов на выборку данных для модели машинного обучения	6
Подключение веб-сервиса к функциональности машинного обучения	8
<b>Примеры запросов на выборку данных для модели машинного обучения</b>	<b>10</b>
Использование утилитного класса Terrasoft.Configuration.QueryExtensions	10
<b>Подключить веб-сервис к функциональности машинного обучения</b>	<b>13</b>
Алгоритм реализации примера	13
<b>Реализовать пользовательскую предиктивную модель</b>	<b>22</b>
Описание кейса	22
Алгоритм выполнения кейса	23
<b>Класс IMLModelTrainerJob</b>	<b>26</b>
Методы	26
<b>Класс IMLModelTrainer</b>	<b>27</b>
Методы	27
<b>Класс IMLServiceProxy</b>	<b>27</b>
Методы	27
Свойства типа ClassificationResult	28
<b>Класс IMLEntityPredictor</b>	<b>29</b>
Методы	29
<b>Класс IMLPredictionSaver</b>	<b>29</b>
Методы	29
<b>Справочник MLModel</b>	<b>29</b>
Поля справочника MLModel	30

# Сервис машинного обучения



## Общие принципы работы сервиса машинного обучения

Сервис машинного обучения (сервис прогнозирования значений справочного поля) использует методы статистического анализа для обучения на основании набора исторических данных. Историческими данными могут быть, например, обращения в службу поддержки за год. При этом используется текст обращения, а также дата и категория контрагента. Результатирующим является поле [ *Группа ответственных* ].

## Схема взаимодействия Creatio с сервисом прогнозирования

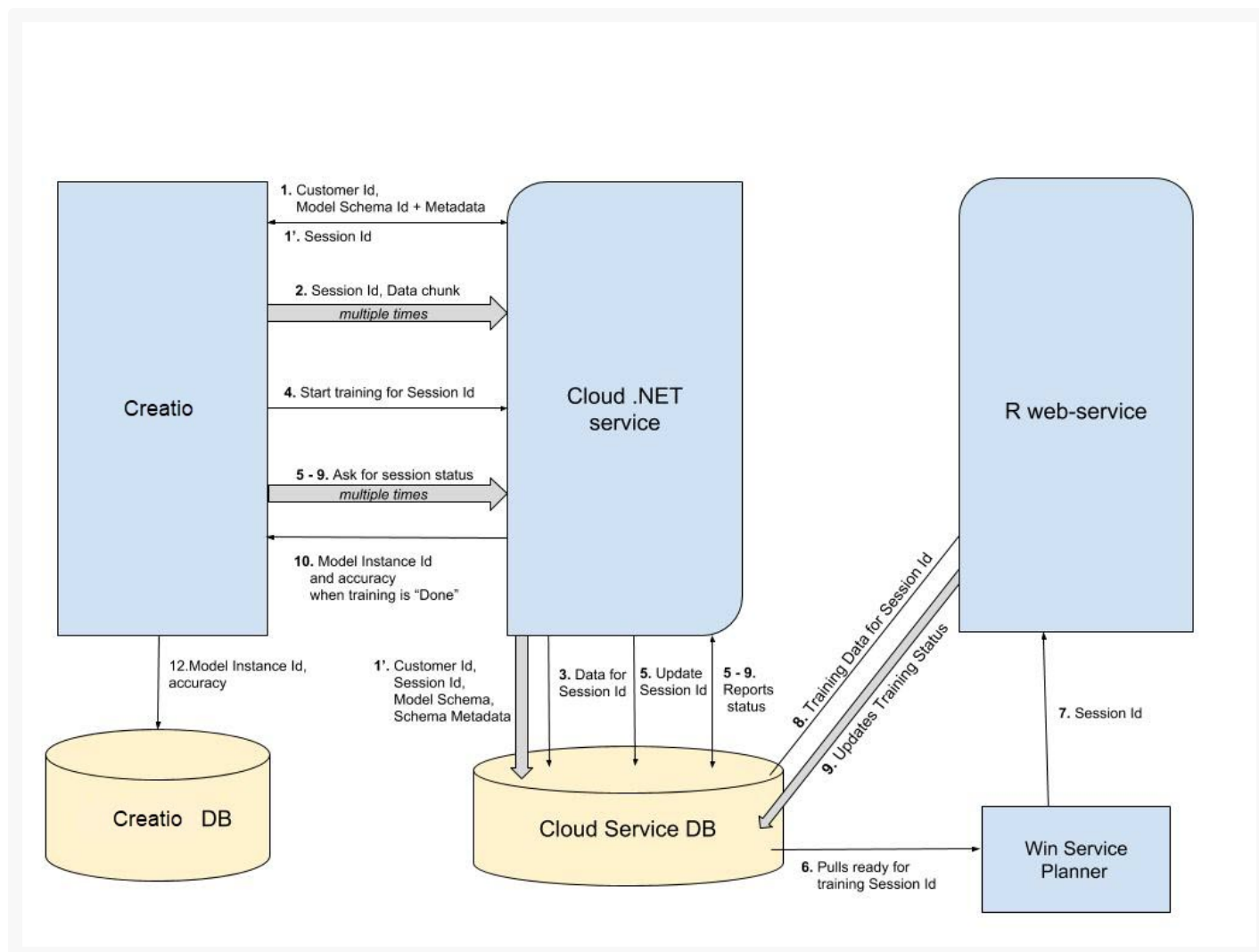
Существует два основных этапа работы Creatio для каждой модели: обучение и прогнозирование.

**Модель прогнозирования** — это алгоритм, который строит прогнозы и позволяет автоматически принимать полезное решение на основе исторических данных.

### Обучение

На этапе обучения выполняется "тренировка" сервиса. Основные шаги обучения:

- Установление сессии передачи данных и обучения.
- Последовательная выборка порции данных для модели и их загрузка в сервис.
- Запрос на постановку в очередь для обучения.
- `Training engine` для обучения модели обрабатывает очередь, обучает модель и сохраняет ее параметры во внутреннее хранилище.
- Creatio периодически опрашивает сервис для получения статуса модели.
- Как только для статуса модели установлено значение `Done` — модель готова для прогнозирования.



## Прогнозирование

Задача прогнозирования выполняется через вызов облачного сервиса с указанием Id экземпляра модели и данных для прогноза. Результат работы сервиса — набор значений с вероятностями, который сохраняется в Creatio в таблице `MLPrediction`.

Если существует прогноз в таблице `MLPrediction` по определенной записи сущности, то на странице записи автоматически отображаются спрогнозированные значения для поля.

Gender

<b>Female</b>	<b>75%</b>
Male	25%

## Настройки и типы данных Creatio для работы с сервисом прогнозирования

## Настройки Creatio

Настройки Creatio, которые необходимо выполнить для работы с сервисом прогнозирования, описаны в статье [Заполнить настройки Creatio](#).

## Расширение логики обучения модели

Описанная выше цепочка классов вызывает и создает экземпляры друг друга посредством IOC контейнера `Terrasoft.Core.Factories.ClassFactory`.

Если необходимо заместить логику какого-либо компонента, то нужно реализовать соответствующий интерфейс. При запуске приложения следует выполнить привязку интерфейса в собственной реализации.

Интерфейсы для расширения логики:

- `IMLModelTrainerJob` — реализация этого интерфейса позволит изменить набор моделей для обучения.
- `IMLModelTrainer` — отвечает за логику загрузки данных для обучения и обновления статуса моделей.
- `IMLServiceProxy` — реализация этого интерфейса позволит выполнять запросы к произвольному предиктивному сервису.

## Вспомогательные классы для прогнозирования

Вспомогательные (утилитные) классы для прогнозирования позволяют реализовать два базовых кейса:

1. Прогнозирование в момент создания или обновления записи какой-либо сущности на сервере.
2. Прогнозирование при изменении сущности на странице записи.

В случае прогнозирования на стороне сервера Creatio — создается бизнес-процесс, который реагирует на сигнал создания/изменения сущности, вычитывает набор полей и вызывает сервис прогнозирования. При получении корректного результата он сохраняет набор значений поля с вероятностями в таблицу `MLClassificationResult`. При необходимости бизнес-процесс записывает отдельное значение (например, с наибольшей вероятностью) в соответствующее поле сущности.

## Составление запросов на выборку данных для модели машинного обучения

Для выборки тренировочных данных или данных для прогнозирования сервиса машинного обучения используется экземпляр класса `Terrasoft.Core.DB.Select`. Он динамически интерпретируется с помощью `Terrasoft.Configuration.ML.QueryInterpreter`.

**Важно.** Интерпретатор `QueryInterpreter` не допускает использования лямбда-выражений.

При составлении выражения запроса в качестве аргумента типа `Terrasoft.Core.UserConnection` в конструкторе `Select` следует использовать предоставляемую переменную `userConnection`. Также обязательным в выражении запроса является наличие колонки с псевдонимом "Id" — уникальным идентификатором экземпляра целевого объекта.

Так как выражение `Select` может быть довольно сложным, то для упрощения его читаемости были добавлены следующие возможности:

- Динамическое добавление типов для интерпретатора.
- Использование локальных переменных.
- Использование утилитного класса `Terrasoft.Configuration.QueryExtensions`.

## Динамическое добавление типов для интерпретатора

Существует возможность динамически добавлять типы для интерпретатора. Для этого класс `QueryInterpreter` предоставляет методы `RegisterConfigurationType` и `RegisterType`. Их можно напрямую использовать в выражении. Например, вместо непосредственного использования идентификатора типа:

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

можно использовать название константы из динамически зарегистрированного перечисления:

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

## Использование локальных переменных

Существует возможность использовать локальные переменные для предотвращения дублирования кода и более удобного структурирования. Ограничение: тип переменной должен быть статически вычисляем и определяется ключевым словом `var`.

Например, запрос с повторяющимся использованием делегатов:

```
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()));
```

можно переписать следующим образом:

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

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

## Подключение веб-сервиса к функциональности машинного обучения

**Важно.** Возможность подключения пользовательского веб-сервиса к функциональности машинного обучения доступна в Creatio версии 7.16.2 и выше.

Задачи машинного обучения (классификация, скоринг, числовая регрессия) или другие подобные задачи (например, прогнозирование оттока клиентов) можно реализовать с помощью веб-сервиса. Эта статья описывает процесс подключения к Creatio пользовательского веб-сервиса, реализующего задачу машинного обучения.

Общий порядок действий при подключении пользовательского веб-сервиса к сервису машинного обучения:

1. Создайте веб-сервис — движок машинного обучения.
2. Расширьте список задач сервиса машинного обучения.
3. Реализуйте модель машинного обучения.

### Создание веб-сервиса — движка машинного обучения

Пользовательский веб-сервис должен реализовать контракт для обучения и выполнения прогнозов по существующей модели. Пример контракта можно посмотреть через Swagger сервиса машинного обучения Creatio.

#### Адрес контракта сервиса машинного обучения Creatio

<https://demo-ml.bpmonline.com/swagger/index.html#/MLService>

Обязательные методы:

- `/session/start` — начало сессии обучения модели.



- `/data/upload` — передача данных в рамках открытой сессии обучения.
- `/session/info/get` — получение информации о состоянии сессии.
- `<пользовательский метод начала обучения>` — метод, который будет вызван Creatio по завершении процесса передачи данных. Процесс обучения модели не должен быть завершен до завершения выполнения этого метода. Процесс обучения может длиться произвольное количество времени (десять минут и даже часы). Когда обучение завершено, метод `/session/info/get` должен будет вернуть состояние сессии обучения `Done` или `Error` в зависимости от результата обучения. Кроме этого, если модель успешно обучена, то необходимо вернуть информацию об экземпляре модели — `ModelSummary`: тип метрики, значение метрики, идентификатор экземпляра и другие.
- `<пользовательский метод прогнозирования>` — метод произвольной сигнатуры, который будет выполнять прогнозирование данных на основании идентификатора экземпляра обученной модели.

Процесс разработки веб-сервиса с использованием IDE Microsoft Visual Studio описан в [статье](#).

## Расширение списка задач сервиса машинного обучения

Для расширения списка задач машинного обучения в Creatio необходимо дополнить справочник `[ MLProblemType ]` новой записью. Необходимо указать следующие параметры:

- `Service endpoint Url` — адрес работающего сервиса машинного обучения.
- `Training endpoint` — путь метода начала обучения.
- `Prediction endpoint` — путь метода прогнозирования.

**Важно.** Для дальнейшей работы необходимо знать идентификатор созданной записи. Посмотреть Id можно в таблице `[dbo.MLProblemType]` базы данных.

## Реализация модели машинного обучения

Для настройки и отображения модели машинного обучения, возможно, потребуется расширить схему карточки `MLModelPage`.

### Реализация IMLPredictor

Необходимо реализовать метод `Predict`, который на вход получит данные, выгруженные из системы по объекту (в виде `Dictionary,>`, где `key` — название поля, а `value` — его значение), а на выход вернет результат прогноза. Метод может использовать прокси-класс, реализующий интерфейс `IMLSERVICEProxy`, который упростит вызов веб-сервиса.

### Реализация IMLEntityPredictor

Необходимо инициализировать свойство `ProblemTypeId` идентификатором созданной записи в справочнике `MLProblemType`. Также необходимо реализовать следующие методы:

- `SaveEntityPredictedValues` — метод получает результат прогнозирования и должен сохранить его для

объекта ( `entity` ) системы, для которого выполнялось прогнозирование. Если возвращаемое значение типа `double` или подобно результату классификации, можно воспользоваться методами вспомогательного класса `PredictionSaver` .

- `SavePrediction` (опционально) — метод сохраняет результат прогнозирования в привязке к экземпляру обученной модели и идентификатору объекта ( `entityId` ). Для базовых задач в системе существуют объекты `MLPrediction` и `MLClassificationResult` .

## Расширение `IMLServiceProxy` и `MLServiceProxy` (опционально)

Можно расширить существующий интерфейс `IMLServiceProxy` и его реализацию методом прогнозирования для текущей задачи. В частности, класс `MLServiceProxy` содержит обобщенный метод `Predict` , который принимает контракты для входящих данных и результата прогнозирования.

## Реализация `IMLBatchPredictor`

В случае, если сервис вызывается с набором данных (500 шт.), следует реализовать интерфейс `IMLBatchPredictor` . Необходимо реализовать следующие методы:

- `FormatValueForSaving` — метод возвращает значение, преобразованное для сохранения результата прогноза в базе данных. В случае пакетного прогнозирования для ускорения операции сохранения запись обновляется в базе данных методом `Update`, а не через экземпляры `Entity`.
- `SavePredictionResult` — определяет, как будет сохраняться результат прогнозирования в системе для каждой записи. Для базовых задач в системе существуют объекты `MLPrediction` и `MLClassificationResult` .

# Примеры запросов на выборку данных для модели машинного обучения



## Использование утилитного класса `Terrasoft.Configuration.QueryExtensions`

Утилитный класс `Terrasoft.Configuration.QueryExtensions` предоставляет несколько расширяющих методов для `Terrasoft.Core.DB.Select` . Это позволяет составлять более компактные запросы.

Для всех расширяющих методов в качестве аргумента `object sourceColumn` могут быть использованы следующие типы (они будут трансформированы в `Terrasoft.Core.DB.QueryColumnExpression` ):

- `System.String` — название колонки в формате "`TableAlias.ColumnName as ColumnAlias`" (где `TableAlias` и `ColumnAlias` опциональны) или "\*" — все колонки.
- `Terrasoft.Core.DB.QueryColumnExpression` — будет добавлен без изменений.
- `Terrasoft.Core.DB.IQueryColumnExpressionConvertible` — будет сконвертировано.
- `Terrasoft.Core.DB.Select` — будет рассмотрен как подзапрос.

**Важно.** Если тип не поддерживается, то будет сгенерировано исключение.

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

Добавляет в запрос указанные колонки или подвыражения.

Используя расширяющий метод `Cols()`, вместо громоздкого выражения:

```
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");
```

МОЖНО ЗАПИСАТЬ:

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

## public static Select Count(this Select select, object sourceColumn)

Добавляет в запрос агрегирующую колонку для вычисления количества непустых значений.

Например, вместо:

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

МОЖНО ЗАПИСАТЬ:

```
var activitiesCount = new Select(userConnection)
```

```
.Count("*") // Здесь можно указать и название колонки.
.From("Activity")
```

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

Добавляет в запрос колонку с функцией определения первого значения, не равного `NULL`.

Например, вместо:

```
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");
```

МОЖНО ЗАПИСАТЬ:

```
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");
```

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

Добавляет в запрос колонку с определением разницы дат.

Например, вместо:

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

можно записать:

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

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

Добавляет в запрос колонку с функцией замены значения `NULL` замещающим выражением.

Например, вместо:

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

можно записать:

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

# Подключить веб-сервис к функциональности машинного обучения

 Средний

**Пример.** Подключить к Creatio пользовательский аналог предиктивного скоринга.

## Алгоритм реализации примера

### 1. Создайте веб-сервис — движок машинного обучения

Пример реализации веб-сервиса машинного обучения для ASP.Net Core 3.1 можно скачать по [ссылке](#).

Реализуйте веб-сервис машинного-обучения `MLService`.

Задайте обязательные методы:

- `/session/start`
- `/data/upload`
- `/session/info/get`
- `/fakeScorer/beginTraining`
- `/fakeScorer/predict`

Полностью исходный код представлен ниже.

#### MLService

```
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 FakeModelInstanceId = "{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.</returns>
[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. Расширьте список задач машинного обучения

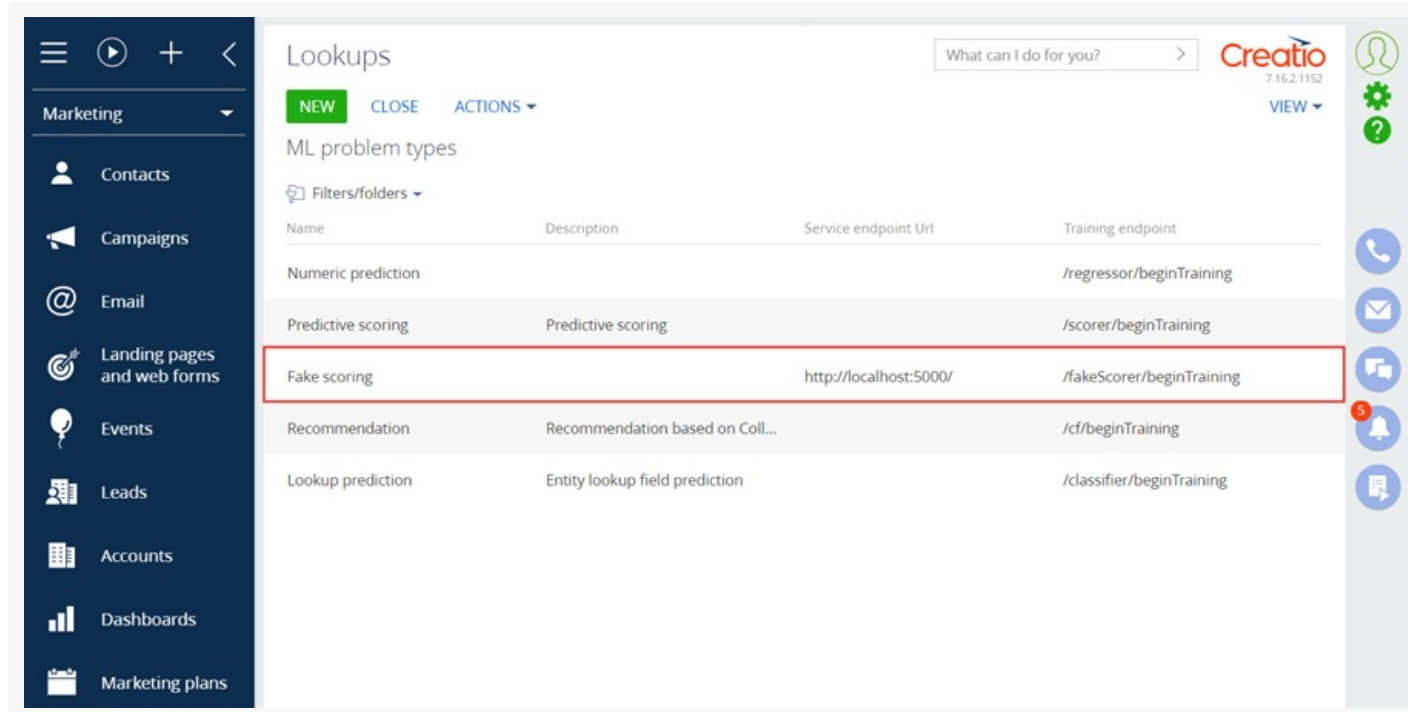
Для расширения списка задач машинного обучения выполните следующие действия:

1. Перейдите в дизайнер системы по кнопке . В блоке [ *Настройка системы* ] ([ *System setup* ]) перейдите по ссылке [ *Справочники* ] ([ *Lookups* ]).
2. Выберите справочник [ *Задачи машинного обучения* ] ([ *ML problem types* ]).
3. Создайте новую запись.

Для записи установите:

- [ *Название* ] ([ *Name* ]) — "Fake scoring";
- [ *Url сервиса* ] ([ *Service endpoint Url* ]) — "http://localhost:5000/";
- [ *Метод сервиса для обучения модели* ] ([ *Training endpoint* ]) — "/fakeScorer/beginTraining".





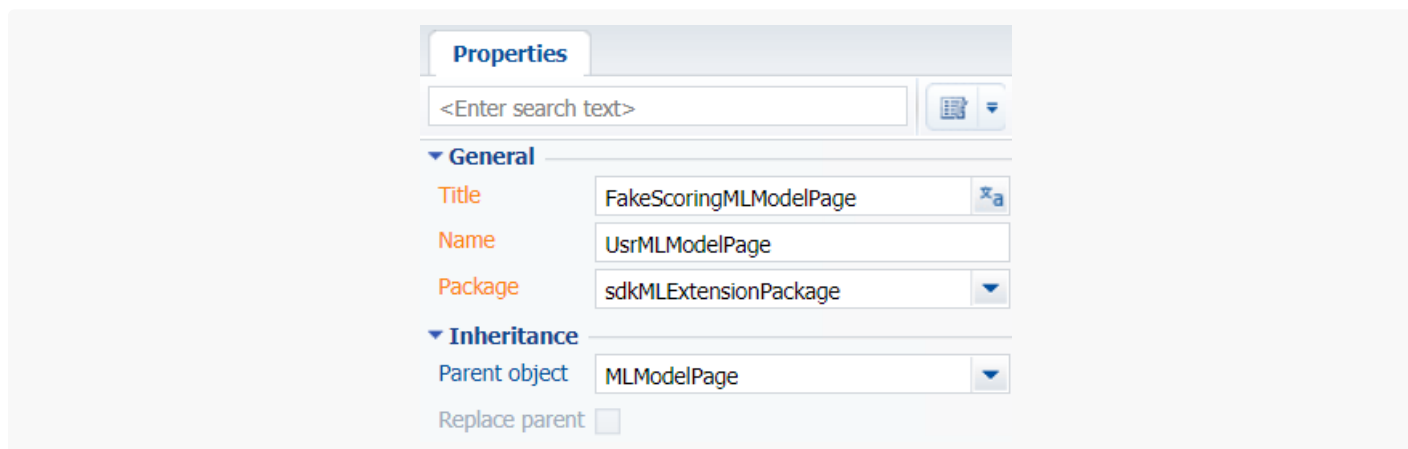
Идентификатор созданной записи — "319c39fd-17a6-453a-bceb-57a398d52636".

### 3. Реализуйте модель машинного обучения

В разделе [ Конфигурация ] ([ Configuration ]) пользовательского пакета на вкладке [ Схемы ] ([ Schemas ]) выполните действие [ Добавить ] —> [ Схема модели представления карточки ] ([ Add ] —> [ Schema of Edit Page View Model ]). Созданный модуль должен наследовать функциональность базовой страницы модели машинного обучения `MLModelPage`, которая определена в пакете `ml`. Для этого укажите эту схему в качестве родительской для создаваемой схемы.

Для создаваемой [схемы объекта](#) установите:

- [ Заголовок ] ([ Title ]) — "FakeScoringMLModelPage";
- [ Название ] ([ Name ]) — "UsrMLModelPage".
- [ Родительский объект ] ([ Parent object ]) — выберите `MLModelPage`.



Переопределите базовый метод `getIsScoring`, чтобы вид создаваемой карточки соответствовал карточке базового предиктивного скоринга. Полностью исходный код представлен ниже.

#### MLModelPage

```
define("MLModelPage", [],
  function() {
    return {
      entitySchemaName: "MLModel",
      properties: {
        FakeScoringProblemTypeId: "319c39fd-17a6-453a-bceb-57a398d52636"
      },
      methods: {
        getIsScoring: function() {
          const parentResult = this.callParent(arguments);
          return parentResult || this.getLookupValue("MLProblemType") === this.FakeSc
        }
      }
    }
  }
});
```

После внесения изменений сохраните и опубликуйте схему.

В разделе [ Конфигурация ] ([ *Configuration* ]) пользовательского пакета на вкладке [ Схемы ] ([ *Schemas* ]) выполните действие [ Добавить ] —> [ Исходный код ] ([ *Add* ] —> [ *Source Code* ]).

Для создаваемой [схемы объекта](#) установите:

- [ Заголовок ] ([ *Title* ]) — "FakeScoringEntityPredictor";
- [ Название ] ([ *Name* ]) — "UsrFakeScoringEntityPredictor".

Реализуйте метод `Predict`, который получает данные, выгруженные из системы по объекту, и возвращает результат прогноза. Метод использует прокси-класс, реализующий интерфейс `IMLServiceProxy`, который упрощает вызов веб-сервиса.

Инициализируйте свойство `ProblemTypeId` идентификатором созданной записи в справочнике `MLProblemType` и реализуйте методы `SaveEntityPredictedValues` и `SavePrediction`. Полностью исходный код представлен ниже.

**FakeScoringEntityPredictor**

```

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(IMPredictor<ScoringOutput>), Name = "319C39FD-17A6-453A-BCEB-57A398D52636")]
    public class FakeScoringEntityPredictor: MLBaseEntityPredictor<ScoringOutput>, IMLEntityPredictor,
        IMPredictor<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, predictedValues);
        }

        protected override void SavePrediction(MLModelConfig model, Guid entityId, ScoringOutput
            predictedResult) {
            PredictionSaver.SavePrediction(model.Id, model.ModelInstanceUid, entityId, predictedResult);
        }
    }
}

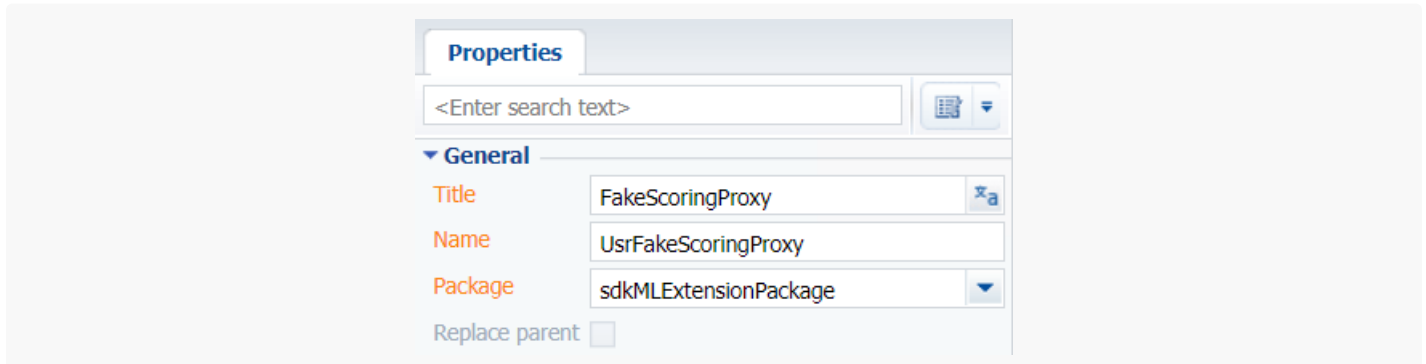
```

После внесения изменений сохраните и опубликуйте схему.

В разделе [ Конфигурация ] ([ Configuration ]) пользовательского пакета на вкладке [ Схемы ] ([ Schemas ]) выполните действие [ Добавить ] —> [ Исходный код ] ([ Add ] —> [ Source Code ]).

Для создаваемой [схемы объекта](#) установите:

- [ Заголовок ] ([ Title ]) — "FakeScoringProxy";
- [ Название ] ([ Name ]) — "UsrFakeScoringProxy".



Расширьте существующий интерфейс `IMLServiceProxy` и его реализацию методом прогнозирования для текущей задачи. В частности класс `MLServiceProxy` содержит обобщенный метод `Predict`, который принимает контракты для входящих данных и результата прогнозирования.

Полностью исходный код представлен ниже.

#### IMLServiceProxy

```
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 predictContributions);
        List<ScoringOutput> FakeScore(MLModelConfig model, IList<Dictionary<string, object>> data, bool predictContributions);
    }

    public partial class MLServiceProxy : IMLServiceProxy
    {
        public ScoringOutput FakeScore(MLModelConfig model, Dictionary<string, object> data, bool predictContributions)
        {
            ScoringResponse response = Predict<ExplainedScoringRequest, ScoringResponse>(model.ModelName,
                new List<Dictionary<string, object>> { data }, model.PredictionEndpoint,
```

```

        100, predictContributions);
    return response.Outputs.FirstOrDefault();
}

public List<ScoringOutput> FakeScore(MLModelConfig model, IList<Dictionary<string, object>
    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;
}
}
}
}

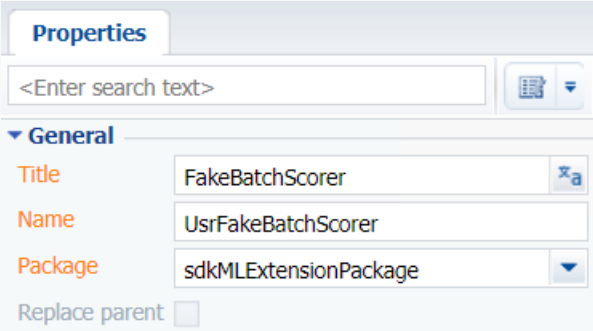
```

После внесения изменений сохраните и опубликуйте схему.

В разделе [ Конфигурация ] ([ *Configuration* ]) пользовательского пакета на вкладке [ Схемы ] ([ *Schemas* ]) выполните действие [ Добавить ] —> [ Исходный код ] ([ *Add* ] —> [ *Source Code* ]).

Для создаваемой [схемы объекта](#) установите:

- [ Заголовок ] ([ *Title* ]) — "FakeBatchScorer";
- [ Название ] ([ *Name* ]) — "UsrFakeBatchScorer".



The screenshot shows the 'Properties' window with the 'General' tab selected. The 'Title' field contains 'FakeBatchScorer', the 'Name' field contains 'UsrFakeBatchScorer', and the 'Package' dropdown menu is set to 'sdkMLExtensionPackage'. There is also a 'Replace parent' checkbox which is currently unchecked.

Для использования функциональности пакетного прогнозирования реализуйте интерфейс

`IMLBatchPredictor`, методы `FormatValueForSaving` и `SavePredictionResult`.

Полностью исходный код представлен ниже.

#### **IMLBatchPredictor**

```

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

```

```
[DefaultBinding(typeof(MLBatchPredictor), 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 modelInstanceId, Guid entityId,
        ScoringOutput value) {
        PredictionSaver.SavePrediction(modelId, modelInstanceId, entityId, value);
    }
}
}
```

После внесения изменений сохраните и опубликуйте схему.

# Реализовать пользовательскую предиктивную модель

 Сложный

## Описание кейса

Реализовать автоматическое прогнозирование колонки [ Категория ] ([ AccountCategory ]) по значениям полей [ Страна ] ([ Country ]), [ Количество сотрудников ] ([ EmployeesNumber ]), [ Отрасль ] ([ Industry ]) во время сохранения записи контрагента. При этом должны выполняться следующие условия:

- Обучение модели необходимо формировать на основании записей о контрагентах за последние 90 дней.
- Переобучение производить каждые 30 дней.
- Допустимое значение точности прогнозирования для модели в целом — 0.6.

### Важно.

Для выполнения этого кейса необходимо удостовериться, что установлено корректное значение для системной настройки [ API ключ облачных сервисов Creatio ] ([ Creatio cloud services API key ], код CloudServicesAPIKey) и установлен адрес предиктивного сервиса в поле [ Url сервиса ] ([ Service endpoint Url ]) записи справочника [ Задачи машинного обучения ] ([ ML problem types ]).

**На заметку.** Описанный кейс можно выполнить, используя UI-инструменты карточки (поля и фильтры), а также элемент бизнес-процесса «Прогнозирование данных». Примеры реализации прогнозирования с помощью встроенных возможностей системы описаны в разделе ["Предиктивный анализ данных"](#).

# Алгоритм выполнения кейса

## 1. Обучение модели

Для обучения модели необходимо:

1. Добавить запись в справочник [ *Модели машинного обучения* ] ( `MLModel` ). Значения полей записи приведены в таблице 1.

Табл. 1. — Значения полей записи справочника MLModel

Поле	Значение
Название	Прогнозирование категории контрагента
Задача машинного обучения	Прогнозирование справочного поля
Идентификатор корневого объекта	Контрагент
Нижний порог допустимого качества	0.6
Частота переобучения	30
Метаданные выборки для обучения	<pre>{   "inputs": [     {       "name": "CountryId",       "type": "Lookup",       "isRequired": true     },     {       "name": "EmployeesNumberId",       "type": "Lookup",       "isRequired": true     }   ] }</pre>

```
},
{
  "name": "IndustryId",
  "type": "Lookup",
  "isRequired": true
}
],
"output": {
  "name": "AccountCategoryId",
  "type": "Lookup",
  "displayName": "AccountCategory"
}
}
```

Выражение для  
выборки данных  
обучения

```
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", "AccountCategoryId")
  .Where("a", "CreatedOn").IsGreater(Column.Parameter(DateTime.Now.AddDays(1)))
```

Примеры составления запросов можно посмотреть в статье [Составление запросов для модели машинного обучения](#).

Флаг,  
включающий  
прогнозирование  
по модели

Отметить

2. Выполнить действие [ *Запланировать задание обучения моделей* ] на странице справочника [ *Модели машинного обучения* ] ( `MLModel` ).

Далее необходимо подождать пока поле [ *Статус обработки моделей* ] пройдет цепочку значений `DataTransfer`, `QueuedToTrain`, `Training`, `Done`. Процесс ожидания может занять несколько часов (зависит от объема передаваемых данных и общей нагрузки на сервис предиктивных моделей).

## 2. Выполнение прогнозов

Для выполнения прогнозов необходимо:



1. В пользовательском пакете создать бизнес-процесс, для которого стартовым сигналом будет событие сохранения объекта [ *Контакт* ]. При этом нужно проверить заполненность значениями необходимых полей (рис. 1).

Рис. 1. — Свойства стартового сигнала

2. В бизнес-процесс добавить параметр-справочник `MLModelId`, ссылающийся на сущность [ *Модель машинного обучения* ]. В качестве значения нужно выбрать запись с созданной моделью [ *Прогнозирование категории Контрагента* ].
3. В бизнес-процесс добавить параметр-справочник `RecordId`, ссылающийся на сущность [ *Контрагент* ]. В качестве значения нужно выбрать ссылку на параметр `RecordId` элемента [ *Сигнал* ].
4. На диаграмму бизнес-процесса добавить элемент [ *Задание-сценарий* ] и добавить в него следующий исходный код:

```
var userConnection = Get<UserConnection>("UserConnection");
// Получение Id записи контрагента
Guid entityId = Get<Guid>("RecordId");
// Получение id модели.
var modelId = Get<Guid>("MLModelId");
var connectionArg = new ConstructorArgument("userConnection", userConnection);
// Объект для вызова прогнозирования
var predictor = ClassFactory.Get<MLEntityPredictor>(connectionArg);
```

```
// Загрузка модели.
// Вызов сервиса прогнозирования. Данные сохраняются в MLPrediction и в случае высокой вероятнос
predictor.PredictEntityValueAndSaveResult(modelId, entityId);
return true;
```

После сохранения и компиляции процесса для новых контрагентов будет выполняться прогнозирование. Полученный прогноз будет отображаться на странице контрагента.

### Важно.

При такой реализации замедляется сохранение отдельной записи контрагента, поскольку вызов сервиса прогнозирования происходит около 2 с. Это может уменьшить производительность массовых операций с сохранением, например при импорте значений из Excel.

## Класс IMLModelTrainerJob C#



Интерфейсы `IJobExecutor`, `IMLModelTrainerJob` используются для задания синхронизации моделей.

Оркестрирует обработку моделей на стороне Creatio, запуская сессии передачи данных, стартуя обучения, а также проверяя статус моделей, обрабатываемых сервисом. Экземпляры по умолчанию запускаются планировщиком заданий через стандартный метод `Execute` интерфейса `IJobExecutor`.

## Методы

`IMLModelTrainerJob.RunTrainer()`

Виртуальный метод, инкапсулирующий логику синхронизации. Базовая реализация этого метода выполняет следующую последовательность действий:

- Выборка моделей для обучения — отбираются записи из `MLModel` по следующему фильтру:
  - поля `MetaData` и `TrainingSetQuery` заполнены.
  - поле `Status` находится в состоянии `NotStarted`, `Done`, `Error` или не заполнено.
  - `TrainFrequency` больше 0.
  - От даты последней попытки обучения (`TriedToTrainOn`) прошло `TrainFrequency` дней.  
Для каждой записи этой выборки производится передача данных в сервис с помощью тренера предиктивной модели (см. ниже).
- Выборка ранее отправленных на обучение моделей и, при необходимости, обновление их статуса.

По каждой подходящей модели начинается сессия передачи данных по выборке. В процессе сессии пакетами по 1000 записей отправляются данные. Для каждой модели объем выборки ограничивается до 75 000 записей.

# Класс IMLModelTrainer



Интерфейс `IMLModelTrainer` является тренером предиктивной модели.

Отвечает за общую обработку отдельно взятой модели на этапе обучения. Коммуникация с сервисом обеспечивается с помощью прокси к предиктивному сервису (см. ниже).

## Методы

---

`IMLModelTrainer.StartTrainSession()`

Устанавливает сессию обучения для модели.

---

`IMLModelTrainer.UploadData()`

Передает данные в соответствии с выборкой модели пакетами по 1000 записей. Выборка ограничивается объемом в 75 000 записей.

---

`IMLModelTrainer.BeginTraining()`

Обозначает завершение передачи данных и сообщает сервису о необходимости постановки модели в очередь обучения.

---

`IMLModelTrainer.UpdateModelState`

Запрашивает у сервиса текущее состояние модели и при необходимости обновляет `Status`.

Если обучение было успешным ( `Status` содержит значение `Done` ), то сервис возвращает метаданные по обученному экземпляру, в частности — точность получившегося экземпляра. Если точность выше или равна нижнему порогу ( `MetricThreshold` ) — идентификатор нового экземпляра записывается в поле `ModelInstanceId`.

# Класс IMLServiceProxy



Интерфейс `IMLServiceProxy` является прокси к предиктивному сервису. Класс-обертка для http-запросов к предиктивному сервису.

## Методы

---

`IMLServiceProxy.UploadData()`

Передаёт пакет данных для сессии обучения.

`IMLServiceProxy.BeginTraining()`

Вызывает сервис постановки обучения в очередь.

`IMLServiceProxy.GetTrainingSessionInfo()`

Запрашивает у сервиса текущее состояние для сессии обучения.

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

Вызывает для ранее обученного экземпляра модели сервис прогнозирования значения поля для отдельно взятого набора значений полей. В параметре `Dictionary data` в качестве ключа передается имя поля, которое обязательно должно совпадать с именем, указанным в поле `MetaData` справочника моделей. При успешном результате метод возвращает список значений типа `ClassificationResult`.

## Свойства типа `ClassificationResult`

Value

Значение поля.

Probability

Вероятность данного значения в диапазоне [0:1]. Сумма вероятностей для одного списка результатов близка к 1 (значения около 0 могут быть опущены).

Significance

Уровень важности данного прогноза.

Возможные значения ( `Significance` )

High	Данное значение поля имеет явное преимущество по сравнению с другими значениями из списка. Такой уровень может быть только у одного элемента в списке предсказанных.
Medium	Значение поля близко к нескольким другим высоким значениям в списке. Например, два значения в списке имеют вероятность 0,41 и 0,39, все остальные значительно меньше.
None	Нерелевантные значения с низкими вероятностями.

# Класс IMLEntityPredictor C#



Утилитный класс, помогающий для отдельной сущности произвести прогнозирование значения поля по указанной модели (одной или несколькими).

## Методы

```
PredictEntityValueAndSaveResult(Guid modelId, Guid entityId)
```

По `Id` модели и `Id` сущности выполняет прогнозирование и записывает результат в результирующее поле объекта. Работает с любой задачей машинного обучения: классификация, скоринг, прогнозирование числового поля.

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

По `Id` модели (или списку нескольких моделей, созданных для одного и того же объекта) и `Id` сущности выполняет классификацию и возвращает словарь, ключ которого — объект модели, значения — список спрогнозированных результатов.

# Класс IMLPredictionSaver C#



Утилитный класс, помогающий сохранить результаты прогнозирования в объект системы.

## Методы

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

Сохраняет результаты классификации ( `MLEntityPredictor.ClassifyEntityValues` ) в объект системы. По умолчанию сохраняет только результат, у которого `Significance` равен “High”. Но есть возможность переопределить это поведение с помощью переданного делегата `onSetEntityValue`. Если делегат вернет `false`, то значение не будет записано в объект системы.

# Справочник MLModel C#



Содержит информацию о выбранных данных для модели, периоде обучения, текущем статусе обучения и т.д.

# Поля справочника MLModel

Назначение основных полей справочника `MLModel` приведено в таблице.

Основные поля справочника `MLModel`

Поле	Тип данных	Назначение
<code>Name</code>	Строка	Название модели.
<code>ModelInstanceUid</code>	Уникальный идентификатор	Идентификатор текущего экземпляра модели.
<code>TrainedOn</code>	Дата/время	Время, когда экземпляр был обучен.
<code>TriedToTrainOn</code>	Дата/время	Время последней попытки обучения.
<code>TrainFrequency</code>	Целое	Частота переобучения модели (дни).
<code>MetaData</code>	Строка	Метаданные с типами колонок выборки. Записываются JSON-формате следующей структуры: <div><div>Структура метаданных</div><pre>{   inputs: [     {       name: "Имя поля 1 в выборке данных",       type: "Text",       isRequired: true     },     {       name: "Имя поля 2 в выборке данных",       type: "Lookup"     },     //...   ],   output: {     name: "Результирующее поле",     type: "Lookup",     displayName: "Имя колонки для отображения"   } }</pre></div>

Здесь:

- `inputs` — набор входящих колонок для модели.
- `output` — колонка, значение которой модель выдает.

- `output` — колонка, значение которой модель должна предсказывать.

В описании колонок поддерживаются следующие атрибуты:

- `name` — имя поля из выражения `TrainingSetQuery`.
- `type` — тип данных для движка обучения.

#### Поддерживаемые значения

Text	Текстовая колонка.
Lookup	Колонка со справочным типом.
Boolean	Логический тип.
Numeric	Числовой тип.
DateTime	Дата и время.

- `isRequired` — обязательность значения поля ( `true` / `false` ). По умолчанию — `false`.

`TrainingSetQuery`

Строка

C#-выражение выборки тренировочных данных. Данное выражение должно возвращать экземпляр класса `Terrasoft.Core.DB.Select`.

#### Пример

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

**Важно.** В выражении для выборки обязательно должна выбираться колонка типа "Уникальный идентификатор". Эта колонка должна называться или иметь псевдоним `Id`.

**Важно.** Если выражение для выборки содержит колонку для сортировки, то эта колонка

обязательно должна присутствовать в результирующей выборке.		
RootSchemaUid	Уникальный идентификатор	Ссылка на схему объекта, для которого будет выполняться прогноз.
Status	Строка	Статус обработки модели (передача данных, обучение, готова для прогноза).
InstanceMetric	Число	Метрика качества для текущего экземпляра модели.
MetricThreshold	Число	Нижний порог допустимого качества модели.
PredictionEnabled	Логическое	Флаг, включающий прогнозирование по данной модели
TrainSessionId	Уникальный идентификатор	Текущая сессия обучения.
MLProblemType	Уникальный идентификатор	Задача машинного обучения (определяет алгоритм и service url, с помощью которого будет обучаться модель