Scalable Batch Predictions
Agenda

1. DataRobot Prediction Options
2. Batch Predictions
3. Sample use case: S3
4. Sample use case: Snowflake
Prediction Options

Test Predictions

- Pre-deployment option for model evaluation on smaller datasets.
- Does not integrate with model monitoring capabilities.

Prediction API

- Synchronous API for low-latency production predictions on deployed models using small datasets.
- Integrates with model monitoring capabilities.

Batch Prediction API

- Asynchronous API for high-volume production predictions on deployed models with the ability to consume and store datasets on cloud object stores and databases.
- Integrates with model monitoring capabilities.
A Batch Prediction Pipeline

Local file

Cloud Storage

Preprocessing

Prediction cluster

Cloud Storage

Cloud Database

Cloud Database
Intake Options

Local file

Stream a CSV file from a local disk through the Batch Prediction API. Not recommended for very large datasets, because of the inherent instability of a data stream across the internet.

Cloud Object Storage

Stream a CSV file stored in AWS S3, Azure Storage or GCP Cloud Storage. Use this for very large datasets. These have suitable APIs and connectivity for achieving very high throughput.

Cloud Databases

Score data stored in Snowflake, Redshift, PostgreSQL, MSSQL or any JDBC capable database. If your data is in a compatible database, you can use our JDBC capabilities to score it without exporting, converting and uploading the data.
Output Options

Local file
After scoring, download the scored data as CSV from the API. Will be removed 48 hours after scoring.

Cloud Object Storage
Saves the scored data back into AWS S3, Azure Storage or GCP Cloud Storage.

Cloud Databases
Create a new table or insert or update scored rows into an existing table.
Predictions Table Format

**Regression**

<TARGET_NAME>_PREDICTION

**Classification**

<TARGET_NAME>_<CLASS_NAME>_PREDICTION
<TARGET_NAME>_PREDICTION
THRESHOLD and POSITIVE_CLASS for binary classification

**Time-Series**

<TIMESTAMP_COLUMN>
FORECAST_POINT and FORECAST_DISTANCE for forecast

**Anomaly Detection**

No target, but: ANOMALY_SCORE
Error handling

**Job level errors**

Errors that abort the job immediately:
- Missing columns/features

**Row level errors**

Errors that only affect the individual row:
- Numeric value wasn’t numeric

**Too many row level errors**

After 20% or 100 MB of row level errors, the job will be aborted.

**Saving the prediction status**

Failed predictions will result in empty predictions being written.
The cause can be added to the output in a special `prediction_status` column.
Column Remapping

Rename columns to match your schema

Supply a mapping with columns to rename:

READMITTED_PREDICTION → readmitted

Remove unneeded columns

Columns mapped to null will be removed from the output
Supported models

Multi-class or Time-Series models does not support prediction explanations yet.

Top N features explained

If explanations are requested for a job, you will get the N most impactful features explained for each prediction.

Representation in the table format

Adds four extra columns per explanation requested:
- EXPLANATION_\(<N>\)_FEATURE_NAME
- EXPLANATION_\(<N>\)_STRENGTH
- EXPLANATION_\(<N>\)_ACTUAL_VALUE
- EXPLANATION_\(<N>\)_QUALITATIVE_STRENGTH
Passthrough Columns

Correlate data with predictions
A primary key or any other ID to identify the prediction

Keep a subset of columns
Send a list in the job parameters

Keep everything
Or annotate the existing dataset with the prediction columns
Scalability Controls

Controlling the number of concurrent prediction requests
Leave resources for other use cases or use everything

Adaptive chunk sizes
The API has heuristics for determining the most effective chunk size when scoring against prediction servers.

Dynamic cluster scaling
Automatically scale to the number of prediction cores allocated to your account or configuration.

Rate limited per cluster
To ensure a prediction cluster is not overloaded, we only run one job at a time per prediction cluster.
Model Monitoring

**Deployment health**

Any predictions from a batch prediction job will be included in the totals for the deployment.

**Data drift tracking**

If enabled for the deployment, the prediction data will be tracked for drift purposes.

Can be disabled per-job.

**Target drift tracking**

If enabled for the deployment, the target will be tracked for drift purposes.

Requires a correlation ID to be passed which will then be added to the passthrough columns automatically.

Can be disabled per-job.
Pinned model during scoring

DataRobot guarantees that a job will be scored using the same model, even if the deployed model changes during scoring.
Life cycle of a job

Initializing
Waiting to receive data or for a processing slot

Running
Job is being scored

Completed
Job completed successfully

Aborted
Job was aborted either by user request or because of too many errors
Accessing the API

Integrate with the REST API
Available in the DataRobot Public API under:
https://app.datarobot.com/api/v2/batchPredictions/

Integrate with the Python SDK
Included in the Python SDK starting with v2.20:
```
dr.BatchPredictionJob.score(...)
dr.BatchPredictionJob.score_to_file(...)
dr.BatchPredictionJob.score_s3(...)
```
Local file intake and output

POST /api/v2/batchPredictions/
Content-Type: application/json

{
    "deploymentId": "5eaacc9a5e6e7636cea5c77d",
    "intakeSettings": {"type": "localFile"},
    "outputSettings": {"type": "localFile"},
}
REST API
streaming data

Stream CSV data to DataRobot

PUT /api/v2/batchPredictions/:id/csvUpload/
Content-Type: text/csv; charset=utf-8

col1,col2,col3
a,b,1
c,d,2

Stream scored data from DataRobot

GET /api/v2/batchPredictions/:id/download/

For best performance, run these concurrently
Python SDK job creation

Local file intake and output

```python
import datarobot as dr

job = dr.BatchPredictionJob.score_to_file(
    '5eaacc9a5e6e7636cea5c77d',
    'prediction_data.csv',
    'predicted.csv',
)

job.wait_for_completion()
```
Sample use case: S3 Scoring

**Private bucket**

- Either securely store credentials in DataRobot
- Or if running a Private AI Cloud in AWS, use an IAM instance profile

**Scalability notes**

- Most performant way to score large CSV files
- Currently limited to 100 GB output on SaaS, but depends on your environment.
Sample use case: S3 Scoring

Example job

credential = dr.Credential.create_s3(
    name='S3',
    aws_access_key_id='...',
    aws_secret_access_key='...',
)

dr.BatchPredictionJob.score_s33(
    '5eaacc9a5e6e7636cea5c77d',
    's3://private-bucket/prediction_data.csv',
    's3://private-bucket/predicted.csv',
    credential=credential,
)

dr.BatchPredictionJob.wait_for_completion()
Sample use case: Snowflake Scoring

Setup data connection in DataRobot
Sample use case: Snowflake Scoring

Credential and Data Store ID

- Retrieve from API

Supported read modes

- Read entire table
- Craft own query

Supported write modes

- Using existing table
  - Insert
  - Update
- Create new table
Sample use case: Snowflake Scoring

Create a table to hold the predictions

Snowflake doesn’t handle single row UPDATE queries well. So we’ll store predictions in a new table and join against it.

CREATE TABLE batch_demo_kickcars_predictions (  
    "refid" NUMBER(38, 0),  
    "y_1_PREDICTION" FLOAT,  
    "y_0_PREDICTION" FLOAT,  
    "y_PREDICTION" VARCHAR(100)  
);
Sample use case: Snowflake Scoring

Example intake configuration

Score the entire table

```python
intake_settings = {
    'type': 'jdbc',
    'data_store_id': '5eaec6a35e6e7629718500e7',
    'credential_id': '5eaec6b85e6e7629718500e8',
    'schema': 'PUBLIC',
    'table': 'BATCH_DEMO_KICKCARS',
}
```
Sample use case: Snowflake Scoring

Example intake configuration

Incremental scoring for new rows in the table

```sql
query = ""
    SELECT * FROM BATCH_DEMO_KICKCARS AS DATA
    LEFT JOIN
        BATCH_DEMO_KICKCARS_PREDICTIONS AS PRED
    ON
        DATA."refid" = PRED."refid"
    WHERE PRED."y_PREDICTION" IS NULL
"""
Sample use case: Snowflake Scoring

Example intake configuration

Updating the intake settings to include a query instead of a table parameter.

```python
intake_settings = {
    'type': 'jdbc',
    'data_store_id': '5eaec6a35e6e7629718500e7',
    'credential_id': '5eaec6b85e6e7629718500e8',
    'schema': 'PUBLIC',
    'query': query,
}
```
Example output configuration

Write predictions to the new table using the INSERT strategy.

```python
output_settings = {
    'type': 'jdbc',
    'data_store_id': '5eaec6a35e6e7629718500e7',
    'credential_id': '5eaec6b85e6e7629718500e8',
    'schema': 'PUBLIC',
    'table': 'BATCH_DEMO_KICKCARS_PREDICTIONS',
    'statement_type': 'insert',
}
```
Sample use case: Snowflake Scoring

**Complete example**

```python
class = dr.BatchPredictionJob.score(
    deployment_id,
    intake_settings=intake_settings,
    output_settings=output_settings,
    passthrough_columns=['refid'],
    column_names_remapping={
        'THRESHOLD': None,
        'POSITIVE_CLASS': None,
    }
)

job.wait_for_completion()
```
Sample use case: Snowflake Scoring

Predictions written to Snowflake

Table updated with predictions

<table>
<thead>
<tr>
<th>Row</th>
<th>refid</th>
<th>y_1_PREDICTION</th>
<th>y_2_PREDICTION</th>
<th>y_3_PREDICTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35318</td>
<td>0.142136288</td>
<td>0.857863712</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>5702</td>
<td>0.495563253</td>
<td>0.569436777</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>28337</td>
<td>0.126443755</td>
<td>0.873556245</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>64927</td>
<td>0.265383905</td>
<td>0.734616095</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>67944</td>
<td>0.588455391</td>
<td>0.415544009</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>31151</td>
<td>0.049790269</td>
<td>0.950209731</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>13689</td>
<td>0.478863474</td>
<td>0.582138535</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>27542</td>
<td>0.452018953</td>
<td>0.547981047</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>41373</td>
<td>0.315109164</td>
<td>0.684890836</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>59735</td>
<td>0.084240399</td>
<td>0.915759601</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>3198</td>
<td>0.143033369</td>
<td>0.856989831</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>29372</td>
<td>0.08466328</td>
<td>0.91553672</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>28234</td>
<td>0.071868373</td>
<td>0.928137027</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>35535</td>
<td>0.387341882</td>
<td>0.81265818</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>28590</td>
<td>0.33648704</td>
<td>0.863512096</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>20157</td>
<td>0.47576515</td>
<td>0.524264497</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>18376</td>
<td>0.519242463</td>
<td>0.480757537</td>
<td>1</td>
</tr>
</tbody>
</table>

Once new rows are added, the job can be re-run and will only score the newly added rows.
Q&A
DataRobot Community

- **Engage, learn, and accelerate** your AI/ML journey
- **Connect** with peers to find solutions to AI challenges
- **Explore** helpful content to take your AI to the next level
- **Build** your brand as an AI expert & thought leader
- **Join** your peers today at community.datarobot.com
- **Questions**: aisuccess-webinars@datarobot.com