Machine Learning Models Deployment using Sklearn2SQL framework

Technical Presentation

Antoine CARME (2022)
Antoine.CARME@outlook.com

https://github.com/antoinecarme/sklearn2sql-demo

What is Sklearn2SQL ?

- sklearn2sql is a “development tool” for generating deployment SQL code from scikit-learn objects.
  - https://github.com/antoinecarme/sklearn2sql-demo
- Using sklearn2sql, it is possible to predict values from an already-fitted classifier or a regressor simply by executing some SQL code.
- It can be seen as an alternative to PMML-based methods to perform In-database processing.
- It performs the deployment where the data are : in the database !!!
  - No data transfer is needed
  - An asset for performance and security.
  - SQL can also be validated and adapted before execution.
  - SQL is software : it can be integrated easily in large systems.
  - Easily distributable : one can generate SQL codes for all databases at the enterprise level and execute the same model with locally-adapted SQL codes.

Supported Machine Learning Models

- It is designed to support all classification and regression methods in scikit-learn
  - SVMs, linear models, naive-bayes. decision trees, MLP, etc
  - Transformers (PCA, imputers, scalers)
  - Feature selection
  - Outlier detection
  - Derived models (random forests, meta-estimators, pipelines, feature unions, ensembles, etc).

Supported Databases

- Support for most popular relational databases has been added progressively. Now, sklearn2sql supports almost all the leading relational databases referenced on DB-Engines.
  - https://db-engines.com/en/ranking/relational+dbms

- Open source databases : PostgreSQL (Just perfect !!!, most derived database), MariaDB (contributed some CTE-related bugs for this project. Very reactive team. All bugs were fixed !!!!
- Commercial databases : Oracle, MS SQL Server, IBM DB2, Teradata (to cover 95% of the market and get real-world tests)
- Embedded databases : SQLite (even in-memory ;). Nice for prototyping, documentation and development. Zero config. Available everywhere (on Android and iOS devices and inside jupyter notebooks ;).
- Hadoop databases : Hive and Impala
- Other : Firebird (low memory footprint. A stress test ;) , Monetdb (columnar, a SQL quality reminder ;)

Sample Codes 1/4

```sql
-- This SQL code was generated by sklearn2sql (development version).
-- Copyright 2018

-- Model : DecisionTreeClassifier
-- Dataset : iris
-- Database : mysql


-- This SQL code can contain one or more statements, to be executed in the order they appear in this file.



-- Model deployment code

WITH `DT_node_lookup` AS
(SELECT `ADS`.`KEY` AS `KEY`, CASE WHEN (`ADS`.`Feature_3` <= 0.800000011920929) THEN 1 ELSE CASE WHEN (`ADS`.`Feature_2` <= 4.850000381469727) THEN CASE WHEN (`ADS`.`Feature_3` <= 1.6500000953674316) THEN 4 ELSE CASE WHEN (`ADS`.`Feature_1` <= 3.0999999046325684) THEN 6 ELSE 7 END END ELSE CASE WHEN (`ADS`.`Feature_3` <= 1.75) THEN CASE WHEN (`ADS`.`Feature_2` <= 5.050000190734863) THEN CASE WHEN (`ADS`.`Feature_1` <= 2.3499999046325684) THEN 11 ELSE 12 END ELSE 13 END ELSE 14 END END END AS node_id_2
FROM iris AS `ADS`),
`DT_node_data` AS
(SELECT `Values`.node_id AS node_id, `Values`.feature AS feature, `Values`.threshold AS threshold, `Values`.count AS count, `Values`.depth AS depth, `Values`.parent_id AS parent_id, `Values`.`Proba_0` AS `Proba_0`, `Values`.`LogProba_0` AS `LogProba_0`, `Values`.`Proba_1` AS `Proba_1`, `Values`.`LogProba_1` AS `LogProba_1`, `Values`.`Proba_2` AS `Proba_2`, `Values`.`LogProba_2` AS `LogProba_2`, `Values`.`Decision` AS `Decision`, `Values`.`DecisionProba` AS `DecisionProba`
FROM (SELECT 1 AS node_id, CAST(NULL AS CHAR(256)) AS feature, NULL AS threshold, 37 AS count, 1 AS depth, 0 AS parent_id, 1.0 AS `Proba_0`, 0.0 AS `LogProba_0`, 0.0 AS `Proba_1`, -1.79769313486231e+308 AS `LogProba_1`, 0.0 AS `Proba_2`, -1.79769313486231e+308 AS `LogProba_2`, 0 AS `Decision`, 1.0 AS `DecisionProba` UNION ALL SELECT 4 AS node_id, CAST(NULL AS CHAR(256)) AS feature, NULL AS threshold, 36 AS count, 3 AS depth, 3 AS parent_id, 0.0 AS `Proba_0`, -1.79769313486231e+308 AS `LogProba_0`, 1.0 AS `Proba_1`, 0.0 AS `LogProba_1`, 0.0 AS `Proba_2`, -1.79769313486231e+308 AS `LogProba_2`, 1 AS `Decision`, 1.0 AS `DecisionProba` UNION ALL SELECT 6 AS node_id, CAST(NULL AS CHAR(256)) AS feature, NULL AS threshold, 2 AS count, 4 AS depth, 5 AS parent_id, 0.0 AS `Proba_0`, -1.79769313486231e+308 AS `LogProba_0`, 0.0 AS `Proba_1`, -1.79769313486231e+308 AS `LogProba_1`, 1.0 AS `Proba_2`, 0.0 AS `LogProba_2`, 2 AS `Decision`, 1.0 AS `DecisionProba` UNION ALL SELECT 7 AS node_id, CAST(NULL AS CHAR(256)) AS feature, NULL AS threshold, 1 AS count, 4 AS depth, 5 AS parent_id, 0.0 AS `Proba_0`, -1.79769313486231e+308 AS `LogProba_0`, 1.0 AS `Proba_1`, 0.0 AS `LogProba_1`, 0.0 AS `Proba_2`, -1.79769313486231e+308 AS `LogProba_2`, 1 AS `Decision`, 1.0 AS `DecisionProba` UNION ALL SELECT 11 AS node_id, CAST(NULL AS CHAR(256)) AS feature, NULL AS threshold, 1 AS count, 5 AS depth, 10 AS parent_id, 0.0 AS `Proba_0`, -1.79769313486231e+308 AS `LogProba_0`, 0.0 AS `Proba_1`, -1.79769313486231e+308 AS `LogProba_1`, 1.0 AS `Proba_2`, 0.0 AS `LogProba_2`, 2 AS `Decision`, 1.0 AS `DecisionProba` UNION ALL SELECT 12 AS node_id, CAST(NULL AS CHAR(256)) AS feature, NULL AS threshold, 3 AS count, 5 AS depth, 10 AS parent_id, 0.0 AS `Proba_0`, -1.79769313486231e+308 AS `LogProba_0`, 1.0 AS `Proba_1`, 0.0 AS `LogProba_1`, 0.0 AS `Proba_2`, -1.79769313486231e+308 AS `LogProba_2`, 1 AS `Decision`, 1.0 AS `DecisionProba` UNION ALL SELECT 13 AS node_id, CAST(NULL AS CHAR(256)) AS feature, NULL AS threshold, 3 AS count, 4 AS depth, 9 AS parent_id, 0.0 AS `Proba_0`, -1.79769313486231e+308 AS `LogProba_0`, 0.0 AS `Proba_1`, -1.79769313486231e+308 AS `LogProba_1`, 1.0 AS `Proba_2`, 0.0 AS `LogProba_2`, 2 AS `Decision`, 1.0 AS `DecisionProba` UNION ALL SELECT 14 AS node_id, CAST(NULL AS CHAR(256)) AS feature, NULL AS threshold, 37 AS count, 3 AS depth, 8 AS parent_id, 0.0 AS `Proba_0`, -1.79769313486231e+308 AS `LogProba_0`, 0.0 AS `Proba_1`, -1.79769313486231e+308 AS `LogProba_1`, 1.0 AS `Proba_2`, 0.0 AS `LogProba_2`, 2 AS `Decision`, 1.0 AS `DecisionProba`) AS `Values`),
`DT_Output` AS
(SELECT `DT_node_lookup`.`KEY` AS `KEY`, `DT_node_lookup`.node_id_2 AS node_id_2, `DT_node_data`.node_id AS node_id, `DT_node_data`.feature AS feature, `DT_node_data`.threshold AS threshold, `DT_node_data`.count AS count, `DT_node_data`.depth AS depth, `DT_node_data`.parent_id AS parent_id, `DT_node_data`.`Proba_0` AS `Proba_0`, `DT_node_data`.`LogProba_0` AS `LogProba_0`, `DT_node_data`.`Proba_1` AS `Proba_1`, `DT_node_data`.`LogProba_1` AS `LogProba_1`, `DT_node_data`.`Proba_2` AS `Proba_2`, `DT_node_data`.`LogProba_2` AS `LogProba_2`, `DT_node_data`.`Decision` AS `Decision`, `DT_node_data`.`DecisionProba` AS `DecisionProba`
FROM `DT_node_lookup` LEFT OUTER JOIN `DT_node_data` ON `DT_node_lookup`.node_id_2 = `DT_node_data`.node_id)
 SELECT `DT_Output`.`KEY` AS `KEY`, NULL AS `Score_0`, NULL AS `Score_1`, NULL AS `Score_2`, `DT_Output`.`Proba_0` AS `Proba_0`, `DT_Output`.`Proba_1` AS `Proba_1`, `DT_Output`.`Proba_2` AS `Proba_2`, `DT_Output`.`LogProba_0` AS `LogProba_0`, `DT_Output`.`LogProba_1` AS `LogProba_1`, `DT_Output`.`LogProba_2` AS `LogProba_2`, `DT_Output`.`Decision` AS `Decision`, `DT_Output`.`DecisionProba` AS `DecisionProba`
FROM `DT_Output`
```

Sample Codes 2/4

https://github.com/antoinecarme/sklearn2sql-demo/blob/master/VeryLargeModelsSupport_temp_tables/XGBClassifier/FourClass_500/pgsql/demo3_XGBClassifier_pgsql.sql

```
-- This SQL code was generated by sklearn2sql (development version).
-- Copyright 2018

-- Model : XGBClassifier
-- Dataset : FourClass_500
-- Database : pgsql


-- This SQL code can contain one or more statements, to be executed in the order they appear in this file.
```

```
-- Code For temporary table TMP_20180417010516_0ECQBE_XGB_B0 part 1/2. Create


CREATE TEMPORARY TABLE "TMP_20180417010516_0ECQBE_XGB_B0" (
        "KEY" BIGINT,
        "Score_0" FLOAT,
        "Score_1" FLOAT,
        "Score_2" FLOAT,
        "Score_3" FLOAT
)

 ON COMMIT PRESERVE ROWS
```

```
arg_max_cte AS
(SELECT score_soft_max."KEY" AS "KEY", score_soft_max."Score_0" AS "Score_0", score_soft_max."Score_1" AS "Score_1", score_soft_max."Score_2" AS "Score_2", scor
FROM score_soft_max LEFT OUTER JOIN (SELECT union_with_max."KEY" AS "KEY_Score", min(union_with_max.class) AS "arg_max_Score"
FROM union_with_max
WHERE union_with_max."Score" >= union_with_max."max_Score" GROUP BY union_with_max."KEY") AS "arg_max_t_Score" ON score_soft_max."KEY" = "arg_max_t_Score"."KEY_
FROM score_soft_max) AS soft_max_comp ON soft_max_comp."KEY_softmax" = "arg_max_t_Score"."KEY_Score")
 SELECT arg_max_cte."KEY" AS "KEY", CAST(NULL AS FLOAT) AS "Score_0", CAST(NULL AS FLOAT) AS "Score_1", CAST(NULL AS FLOAT) AS "Score_2", CAST(NULL AS FLOAT) AS
FROM arg_max_cte
```

Sample Codes 3/4

```
-- This SQL code was generated by sklearn2sql (development version).
-- Copyright 2018

-- Model : GaussianNB_Pipeline
-- Dataset : BinaryClass_10
-- Database : oracle


-- This SQL code can contain one or more statements, to be executed in the order they appear in this file.
```

```
-- Code For temporary table 22737_HHC6Q9_ADS_IMP_1_OUT part 1/2. Create


CREATE GLOBAL TEMPORARY TABLE "22737_HHC6Q9_ADS_IMP_1_OUT" (
        "KEY" NUMBER(19) NOT NULL,
        impter_2 BINARY_DOUBLE,
        impter_3 BINARY_DOUBLE,
        impter_4 BINARY_DOUBLE,
        impter_5 BINARY_DOUBLE,
        impter_6 BINARY_DOUBLE,
        impter_7 BINARY_DOUBLE,
        impter_8 BINARY_DOUBLE,
        impter_9 BINARY_DOUBLE,
        impter_10 BINARY_DOUBLE,
        impter_11 BINARY_DOUBLE,
        PRIMARY KEY ("KEY")
)
```

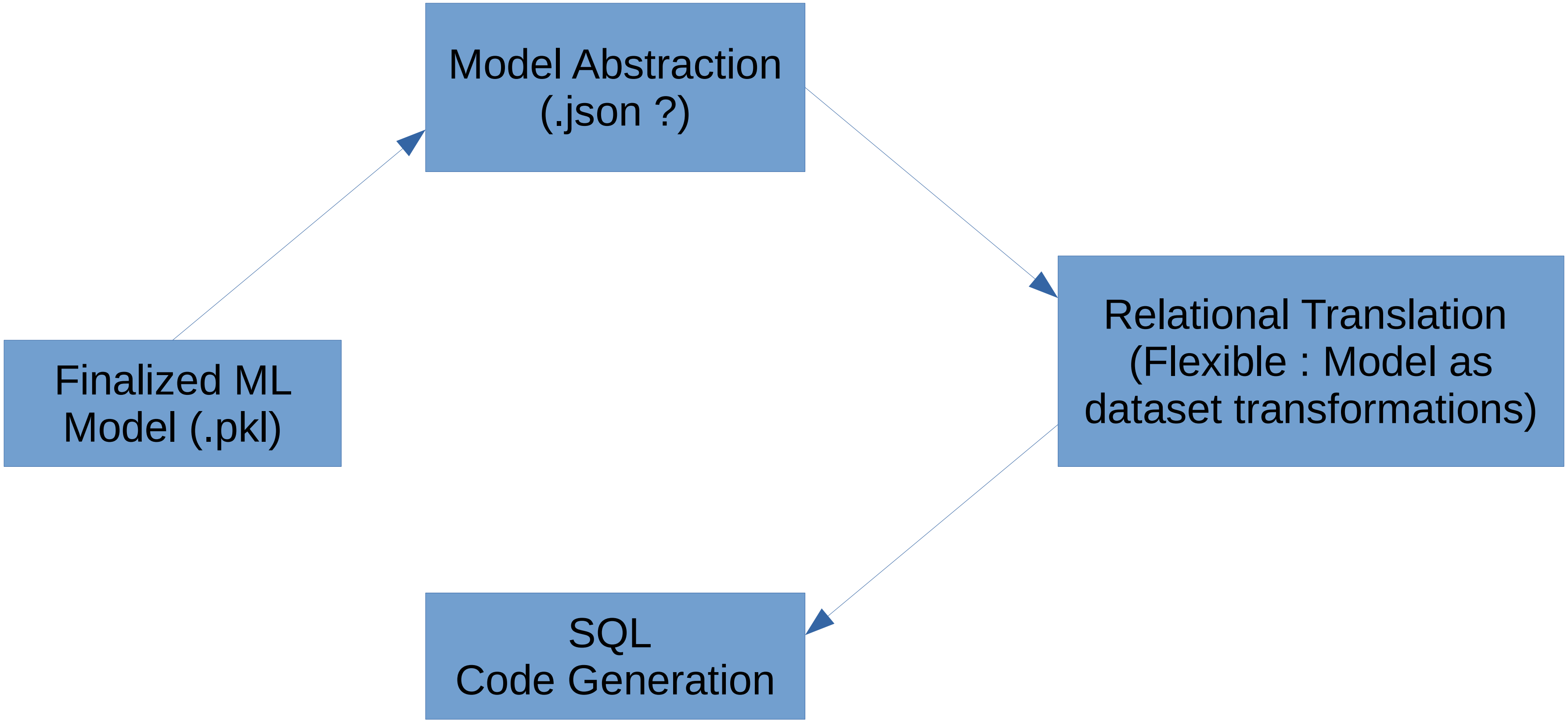
```
-- Code For temporary table 22737_HHC6Q9_ADS_IMP_1_OUT part 2/2. Populate

INSERT INTO "22737_HHC6Q9_ADS_IMP_1_OUT" ("KEY", impter_2, impter_3, impter_4, impter_5, impter_6, impter_7, impter_8, impter_9, impter_10, impter_11) SELECT "U
FROM (SELECT "ADS_imp_1_OUT"."KEY", "ADS_imp_1_OUT".impter_2, "ADS_imp_1_OUT".impter_3, "ADS_imp_1_OUT".impter_4, "ADS_imp_1_OUT".impter_5, "ADS_imp_1_OUT".impt
FROM (SELECT "ADS"."KEY" AS "KEY", CASE WHEN ("ADS"."Feature_0" IS NULL) THEN 0.061829205238134496 ELSE "ADS"."Feature_0" END AS impter_2, CASE WHEN ("ADS"."Fea
FROM "BINARYCLASS_10" "ADS") "ADS_imp_1_OUT") "U"
```

```
arg_max_cte AS
(SELECT score_soft_max."KEY" AS "KEY", score_soft_max."Score_0" AS "Score_0", score_soft_max."Score_1" AS "Score_1", score_soft_max."Proba_0" AS "Proba_0", scor
FROM score_soft_max LEFT OUTER JOIN (SELECT union_with_max."KEY" AS "KEY_Score", min(union_with_max.class) AS "arg_max_Score"
FROM union_with_max
WHERE union_with_max."max_Score" <= union_with_max."Score" GROUP BY union_with_max."KEY") "arg_max_t_Score" ON score_soft_max."KEY" = "arg_max_t_Score"."KEY_Sco
FROM score_soft_max) soft_max_comp ON soft_max_comp."KEY_softmax" = "arg_max_t_Score"."KEY_Score")
 SELECT arg_max_cte."KEY" AS "KEY", CAST(NULL AS BINARY_DOUBLE) AS "Score_0", CAST(NULL AS BINARY_DOUBLE) AS "Score_1", arg_max_cte."SoftProba_0" AS "Proba_0",
FROM arg_max_cte
```

Sample Codes 4/4

```sql
-- This SQL code was generated by sklearn2sql (development version).
-- Copyright 2018

-- Model : SVR_rbf
-- Dataset : boston
-- Database : sqlite


-- This SQL code can contain one or more statements, to be executed in the order they appear in this file.



-- Model deployment code

WITH kernel_input AS
(SELECT "ADS"."KEY" AS "KEY", CAST("ADS"."Feature_0" AS FLOAT) AS "Feature_0", CAST("ADS"."Feature_1" AS FLOAT) AS "Feature_1", CAST("ADS"."Feature_2" AS FLOAT)
FROM boston AS "ADS"),
"SV_data" AS
(SELECT "Values".sv_idx AS sv_idx, CAST("Values".dual_coeff AS FLOAT) AS dual_coeff, CAST("Values".sv_0 AS FLOAT) AS sv_0, CAST("Values".sv_1 AS FLOAT) AS sv_1,
FROM (SELECT 0 AS sv_idx, 0.1 AS dual_coeff, 0.11425 AS sv_0, 0.0 AS sv_1, 13.89 AS sv_2, 1.0 AS sv_3, 0.55 AS sv_4, 6.373 AS sv_5, 92.4 AS sv_6, 3.3633 AS sv_7
kernel_dp AS
(SELECT t."KEY" AS "KEY", t.dot_product AS dot_product
FROM (SELECT full_join_data_sv."KEY" AS "KEY", sum(CAST(full_join_data_sv.dot_prod1 AS FLOAT)) + 21.35026686098341 AS dot_product
FROM (SELECT kernel_input."KEY" AS "KEY", "SV_data".dual_coeff * exp(min(max(-100.0, -0.07692307692307693 * (power(kernel_input."Feature_0" - "SV_data".sv_0, 2)
FROM kernel_input, "SV_data") AS full_join_data_sv GROUP BY full_join_data_sv."KEY") AS t)
 SELECT kernel_dp."KEY" AS "KEY", kernel_dp.dot_product AS "Estimator"
FROM kernel_dp
```

Framework Description



System Input

- The system generates SQL code from already trained models.
- The gory details of the training process are not significant.
- Almost all public interfaces and web services take a serialized model as input (pickle is your friend here).

Model Abstraction

- In this step, the system performs a complete formal abstraction of the underlying algorithm of the model.
  - For a linear Model :
    - {“Type” : “linear”, “coefficients” : [8.88, 8888e-8], “intercept” : 888888}
- More complex formal abstractions are available for each model/algorithm type (DT, XGB, SVM, RF, Pipeline, ...).
- The abstraction must be complete, For a random forest (RF) model, the abstraction contains the individual abstractions of all the forest decision trees.
- Adding a new mathematical model requires designing/authoring these abstractions.
  - The math behind scikit-learn algorithms (and machine learning in general) is evolving slowly (need many years to “invent” a new model). Almost all of these models date back to 19xx.
    - Scikit-learn has a lot of requirements on what model can be added (popularity, publications, citations, impact,), and that’s really good. https://scikit-learn.org/stable/faq.html
  - The current version of the sklearn2sql framework has designed (> 40) abstractions of almost all scikit-learn models known before 2020.
    - https://scikit-learn.org/stable/modules/classes.html
- These abstractions are deduced from a reloaded/living model. We only use model metadata. The original dataset is not needed even if the model can help generating/simulating one.
- The formal abstraction is expressed in JSON format.
  - Almost all python objects have a __dict__ member. Python object introspection is easy.
  - Scikit Learn Models are aggregates of numpy python objects.
  - XGB models already have a to_json method. Easy.
  - JSON code is debuggable.

Relational Model Representation

- All machine learning data processing can be performed using sequences of simple dataset/columns transformations (dataset→dataset mapping).
- This translation is performed only using the “Complete” Model Abstraction (JSON format is enough).
- These dataset transformations can be translated into relations (as in a relational algebra / SQL).
- For a given model, The number of these transformations depends on the complexity of the model.
  - Think of these as the layers of a sequential neural network (Scikit MLP), but at a lower level.
- Individual transformations are developed using specialized python classes, with the most tasks done at the abstract level.
  - The transformation does not take into account the type of model it is part of.
  - Groups of transformations used to compute the predicted values (arg-max of probabilities) are shared across all model representations (the same SQL code between DTs and pipelines of SVMs).
- A scikit-learn machine learning model can be mapped this way.
  - A decision tree score computation can be performed by hand using ~4 excel sheets, and these sheets can be mapped to SQL tables/views. The first table contains the input and the last contains the scores and class probabilities as columns.
  - A random forest with 500 trees, will be translated as ~4*500 separate relations. An additional relation is used to compute the RF class probabilities (means of 500 individual class probabilities).
    - Seems too mechanical. There are some technical limitations but SQL allows the usage of views or temporary tables to deal with this kind of complexity.

SQL Code Generation 1/4

- SQL Code generation is about translation the model representation into a valid SQL code for a target database dialect.
- We use the excellent Python SQLAlchemy as a framework for handling relational model represention internal transformations.
  - https://www.sqlalchemy.org/
- SQLAlchemy allows mapping each transformation to a set of SQLAlchemy objects (table, expression, column, function, join, literal values, case, union, ...)
- SQLAlchemy Expressions can generate their SQL Code for a target database which has a python SQLAlchemy driver.
- SQLAlchemy has a set of internally supported database drivers (psql, sqlite, mssql, oracle, …)
- Almost all database vendors maintain a python SQLAlchemy driver for their database (teradata, ...).
  - Easy to add a new database.
  - For teradata : https://github.com/Teradata/sqlalchemy-teradata
- SQLAlchemy plays nicely with pandas
  - One can create a pandas dataframe using a SQLAlchemy connection and a SQL select statement to be executed.
    - https://pandas.pydata.org/docs/reference/api/pandas.read_sql.html#pandas.read_sql
  - One can save a dataframe as a table through a SQLAlchemy database connection.
    - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_sql.html#pandas.DataFrame.to_sql

SQL Code Generation 2/4

- Sketch of a linear regression model SQL code generation

```
import sqlalchemy as sa
input_cte = sa.Table("boston")
scal_prod_expr = input_cte.columns["Feature1"] * sa.literal_value(coefs[1]) + ...
scal_prod_expr = scal_prod_expr + input_cte.columns["Feature4"] * sa.literal_value(coefs[4])
scal_prod_expr = scal_prod_expr + sa.literal_value(intercept)
estimator = sa.cast_as_float(scal_prod_expr)
output_cte = sa.cte([ input_cte.columns["KEY"] , estimator.label("Estimator") ])
select_est = sa.select(output_cte)
# All the code above is database-agnostic,
lSQL = select_est.generate_sql(psql_connection)
```

SQL Code Generation 3/4

- By default , each transformation is generated as a CTE through an sa.CTE object (WITH clause)
  - https://modern-sql.com/feature/with
- For a simple model, a single select statement which contains one or many CTEs is generated.
  - SQL Code = [With Clauses + compute_scores_select]
- Some databases do not support too many WITH clauses in the same select
  - MariaDB, number of CTEs > 64.
    - https://jira.mariadb.org/browse/MDEV-13730
- For complex models (number of CTEs > 64), the SQL generator materializes some CTEs
  - For a RF model with 500 trees, each tree is generated as temporary table instead of a CTE. These temporary tables are populated on each code execution and automatically dropped by internal database mechanisms.
    - SQL Codes = [500*create_temp_tables , 500*populate_temp_tables , 1*aggregate_scores_select]

SQL Code Generation 4/4

- For the SQL code design, the author tried when possible to use meaningful aliases in the SQL code. The names are intention revealing according to the excellent Modern SQL.
  - http://modern-sql.com/use-case/literate-sql

- For example, a SQL generated from a decision tree contains three common table expressions (CTEs) whose names are : "DT_node_lookup", "DT_node_data" and "DT_Output.
  - https://github.com/antoinecarme/sklearn2sql-demo/blob/master/sample_outputs_tuning_round_1/DecisionTreeClassifier/iris/pgsql/demo2_DecisionTreeClassifier_pgsql.sql

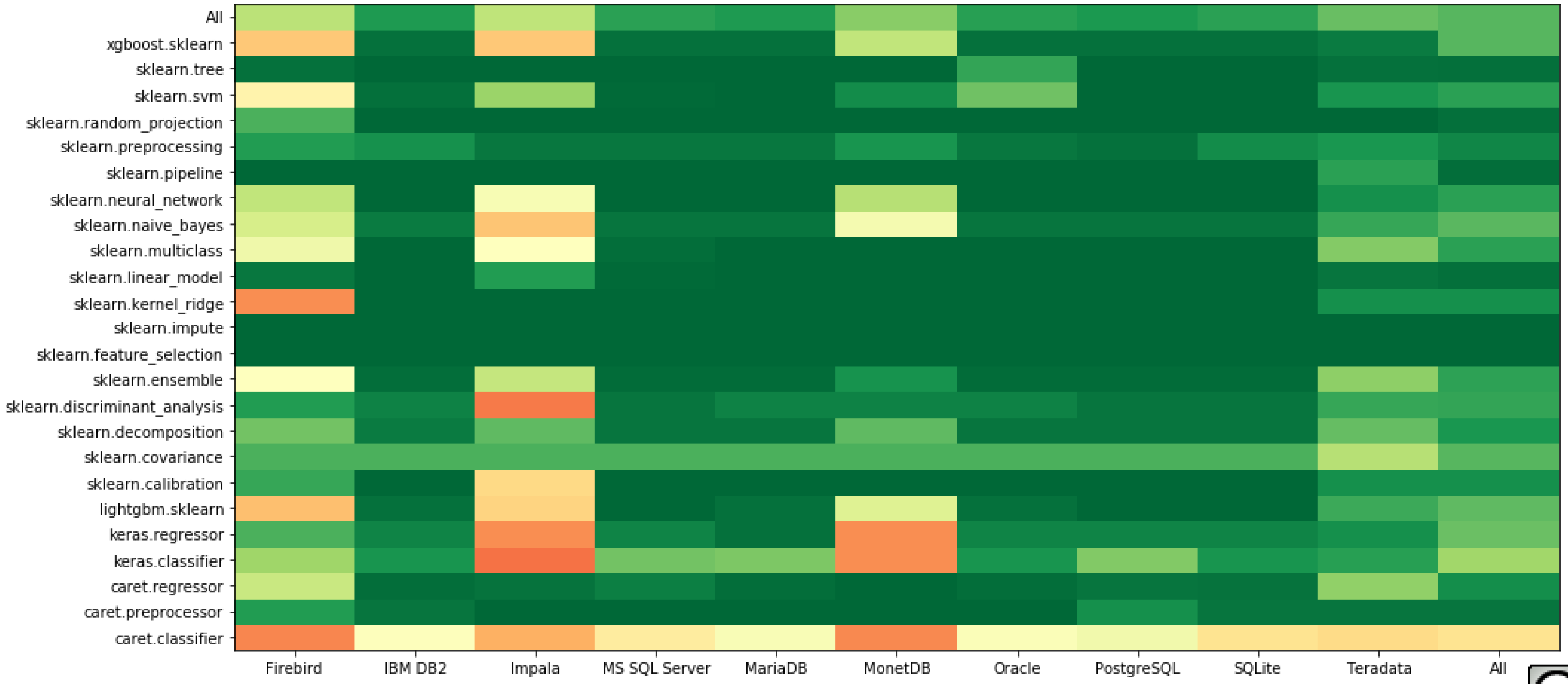
- This design has a significant impact on the performance. All SQL codes are tested to execute in less than 2-3 minutes on the various databases, even for very complex models.

Quality Assessment 1/2

- A test framework has been put in place.
- The framework works on all supported databases instances simultaneously.
- For each scikit-learn model class, we perform the following tests:
  - Train different models on datasets of various sizes (nrows, ncols). These datasets are transferred
  - For each trained model :
    - save it,
    - Predict all the possible values for all the training dataset in python scikit-learn (sklearn_output_python_df)
    - generate the SQL,
    - Execute the SQL code on each database => pandas dataframes (gen_sql_output_mssql_df , ...).
    - Compare the relevant output columns between sklearn_output_python_df and each database output dataframe.
    - Add some reporting for success/failures each model/database combination.

Quality Assessment 2/2

- Sample report :
  - https://github.com/antoinecarme/sklearn2sql_heroku/blob/master/Quality/extensive_tests-debrief.ipynb



Extensions

- The system has been designed to been extended to support additional types of models:
  - Gradient Boosting Models : XGBoost and LightGBM
    - Have a scikit-learn API.
    - These can be exported in a json format.
  - Keras and PyTorch Deep Learning Models
    - Both share the same abstraction layers.
      - https://github.com/antoinecarme/keras2sql
      - https://github.com/antoinecarme/pytorch2sql
    - Defined new transformations for NN Layers and transfer functions (ReLU, ...).
    - Basically he same as sklearn.MLP
    - Support Convolutional networks (partial) and recursive models (RNN, LSTM, GRU)
  - R Models, through Caret.
    - Basically, these are the same as the existing scikit-learn models
      - https://github.com/antoinecarme/caret2sql
    - We chose to generate code for “equivalent” abstractions for scikit-learn models.
    - Abstract models are generated by running custom R model introspection scripts.
  - The four NN dense layer implementations (sklearn.MLP, Keras.Dense, PyTorch.Dense, Caret.nnet) share the same abstraction and the model transformations.
  - These models have been included in the Quality Assessment test framework.
    - R models output (using R) is compared with the generated SQL output for each database.

谢谢 !!!!