

### An MLIR Dialect for Data Analytics

### Arc

PLDS' 20 Workshop March 5, 2020

Presented by **Klas Segeljakt** <<u>klasseg@kth.se</u>>

Joint work with **Frej Drejhammar** <<u>frej.drejhammar@ri.se</u>> **Khoa Dinh** <<u>khoad@kth.se</u>>





- Outline
  - Introduction & Problem
  - Related Work
  - Arc & MLIR
  - Summary



#### KTH VETENSKAP OCH KONST

### Introduction & Problem













[Q1]: What does a programming language for data analytics definitely need?







# [A1]: Data types

[Q1]: What does a programming language for data analytics definitely need?







# [A1]: Data types

[Q2]: Which data types?

**[Q1]:** What does a programming language for data analytics definitely need?







[Q1]: What does a programming language for data analytics definitely need?[A1]: Data types

[Q2]: Which data types? [A2]: Tables, Arrays, Streams, Graphs, ...







[Q1]: What does a programming language for data analytics definitely need?[A1]: Data types

[Q2]: Which data types? [A2]: Tables, Arrays, Streams, Graphs, ...

[Q3]: Can [A1] be "combined" under one system and programming model?





#### KTH VETENSKAP OCH KONST

### Introduction & Problem



4





Optimise





4





Front-end Domain Specific Language (DSL) A sepresentation (IR)

### Optimise

### Execute

Back-end Distributed Runtime







#### **Challenges:**

**Challenges:** 







#### **Challenges:**

#### **Challenges:**

- Abstraction level
- Syntax & type system
- Tooling & Integration







#### **Challenges:**

- Abstraction level
- Syntax & type system
- Tooling & Integration

### **Challenges:**

- Progressive lowering
- Algebraic simplification
- Instruction selection
- Cost model







#### **Challenges:**

- Abstraction level
- Syntax & type system
- Tooling & Integration

#### **Challenges:**

- Progressive lowering • Algebraic simplification Instruction selection
- Cost model

- State management
- Scheduling
- Failure recovery
- Profiling & Debugging































































### What is Lara?







### **An Intermediate Representation for Optimizing Machine Learning Pipelines**

- Andreas Kunft<sup>\*</sup> Asterios Katsifodimos<sup>\*\*</sup> Sebastian Schelter<sup>†</sup> Volker Markl<sup>‡\*</sup> Sebastian Breß<sup>\*\*</sup> Tilmann Rabl<sup>+</sup>
- \*TU Berlin \*\*Delft University of Technology \*New York University \*DFKI +HPI, Universität Potsdam

### What is Lara?































**Problem?** Current systems used for ML are tailored to either

processing (Spark, Flink, etc.) or training (TensorFlow, SystemML, etc.)







**Problem?** Current systems used for ML are tailored to either

➡No holistic optimisation

processing (Spark, Flink, etc.) or training (TensorFlow, SystemML, etc.)







**Problem?** Current systems used for ML are tailored to either **processing** (Spark, Flink, etc.) or **training** (TensorFlow, SystemML, etc.)

➡No holistic optimisation

Solution? Lara: A language for Linear Algebra and Relational Algebra







8



Lara has a Quotation Based DSL embedded in Scala.



8



Lara has a Quotation Based DSL embedded in Scala.

**Types** (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$



8



Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set ( $\mathbb{N}$ , A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$

#### **Operations**:

- For-comprehensions
- Linear algebra functions



8


Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$

#### **Operations**:

- For-comprehensions
- Linear algebra functions

#### Type conversions track provenance:

- DataBag ( $\mathbb{N}$ , Vector A)  $\rightarrow$  Matrix A
- DataBag  $A \rightarrow (A \rightarrow (N, N)) \rightarrow Matrix A$



8



Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$

#### **Operations**:

- For-comprehensions
- Linear algebra functions

Type conversions track provenance:

- DataBag ( $\mathbb{N}$ , Vector A)  $\rightarrow$  Matrix A
- DataBag A  $\rightarrow$  (A  $\rightarrow$  (N, N))  $\rightarrow$  Matrix A



DataBag ( $\mathbb{N}$ , Vector A)



8



Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$

#### **Operations**:

- For-comprehensions
- Linear algebra functions

#### Type conversions track provenance:

- DataBag ( $\mathbb{N}$ , Vector A)  $\rightarrow$  Matrix A
- DataBag A  $\rightarrow$  (A  $\rightarrow$  (N, N))  $\rightarrow$  Matrix A





8



Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$

#### **Operations**:

- For-comprehensions
- Linear algebra functions

#### Type conversions track provenance:

- DataBag ( $\mathbb{N}$ , Vector A)  $\rightarrow$  Matrix A
- DataBag A  $\rightarrow$  (A  $\rightarrow$  (N, N))  $\rightarrow$  Matrix A



Lara also has an IR with two "views".



8



Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$

#### **Operations**:

- For-comprehensions
- Linear algebra functions

#### Type conversions track provenance:

- DataBag ( $\mathbb{N}$ , Vector A)  $\rightarrow$  Matrix A
- DataBag A  $\rightarrow$  (A  $\rightarrow$  (N, N))  $\rightarrow$  Matrix A



Lara also has an IR with two "views".

A Monadic View that enables Dataflow optimisations









Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$

#### **Operations**:

- For-comprehensions
- Linear algebra functions

#### Type conversions track provenance:

- DataBag ( $\mathbb{N}$ , Vector A)  $\rightarrow$  Matrix A
- DataBag A  $\rightarrow$  (A  $\rightarrow$  (N, N))  $\rightarrow$  Matrix A



Lara also has an IR with two "views".

A Monadic View that enables Dataflow optimisations → Operator Fusion and Operator Pushdown









Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$

#### **Operations**:

- For-comprehensions
- Linear algebra functions

#### Type conversions track provenance:

- DataBag ( $\mathbb{N}$ , Vector A)  $\rightarrow$  Matrix A
- DataBag A  $\rightarrow$  (A  $\rightarrow$  (N, N))  $\rightarrow$  Matrix A



Lara also has an IR with two "views".

A Monadic View that enables Dataflow optimisations → Operator Fusion and Operator Pushdown

 $\mathbf{x} + \mathbf{y}$ 

**Element-wise vector addition** 









Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$

#### **Operations**:

- For-comprehensions
- Linear algebra functions

#### Type conversions track provenance:

- DataBag ( $\mathbb{N}$ , Vector A)  $\rightarrow$  Matrix A
- DataBag A  $\rightarrow$  (A  $\rightarrow$  (N, N))  $\rightarrow$  Matrix A



Lara also has an IR with two "views".

A Monadic View that enables Dataflow optimisations → Operator Fusion and Operator Pushdown



Element-wise vector addition









Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$

#### **Operations**:

- For-comprehensions
- Linear algebra functions

#### Type conversions track provenance:

- DataBag ( $\mathbb{N}$ , Vector A)  $\rightarrow$  Matrix A
- DataBag A  $\rightarrow$  (A  $\rightarrow$  (N, N))  $\rightarrow$  Matrix A



Lara also has an IR with two "views".

A Monadic View that enables Dataflow optimisations → Operator Fusion and Operator Pushdown











Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$

#### **Operations**:

- For-comprehensions
- Linear algebra functions

#### Type conversions track provenance:

- DataBag ( $\mathbb{N}$ , Vector A)  $\rightarrow$  Matrix A
- DataBag A  $\rightarrow$  (A  $\rightarrow$  (N, N))  $\rightarrow$  Matrix A



Lara also has an IR with two "views".

A Monadic View that enables Dataflow optimisations → Operator Fusion and Operator Pushdown



A Combinator View that enables Domain-specific opts.







Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$

#### **Operations**:

- For-comprehensions
- Linear algebra functions

#### Type conversions track provenance:

- DataBag ( $\mathbb{N}$ , Vector A)  $\rightarrow$  Matrix A
- DataBag A  $\rightarrow$  (A  $\rightarrow$  (N, N))  $\rightarrow$  Matrix A



Lara also has an IR with two "views".

A Monadic View that enables Dataflow optimisations → Operator Fusion and Operator Pushdown









Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$

#### **Operations**:

- For-comprehensions
- Linear algebra functions

#### Type conversions track provenance:

- DataBag ( $\mathbb{N}$ , Vector A)  $\rightarrow$  Matrix A
- DataBag A  $\rightarrow$  (A  $\rightarrow$  (N, N))  $\rightarrow$  Matrix A



Lara also has an IR with two "views".

A Monadic View that enables Dataflow optimisations → Operator Fusion and Operator Pushdown



A Combinator View that enables Domain-specific opts. Instruction selection and Linear Algebra rewrites



Gram matrix calculation







Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$

#### **Operations**:

- For-comprehensions
- Linear algebra functions

#### Type conversions track provenance:

- DataBag ( $\mathbb{N}$ , Vector A)  $\rightarrow$  Matrix A
- DataBag A  $\rightarrow$  (A  $\rightarrow$  (N, N))  $\rightarrow$  Matrix A



Lara also has an IR with two "views".

A Monadic View that enables Dataflow optimisations → Operator Fusion and Operator Pushdown



A Combinator View that enables Domain-specific opts. Instruction selection and Linear Algebra rewrites



Gram matrix calculation







Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$

#### **Operations**:

- For-comprehensions
- Linear algebra functions

#### Type conversions track provenance:

- DataBag ( $\mathbb{N}$ , Vector A)  $\rightarrow$  Matrix A
- DataBag A  $\rightarrow$  (A  $\rightarrow$  (N, N))  $\rightarrow$  Matrix A



Lara also has an IR with two "views".

A Monadic View that enables Dataflow optimisations → Operator Fusion and Operator Pushdown











Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$

#### **Operations**:

- For-comprehensions
- Linear algebra functions

#### Type conversions track provenance:

- DataBag ( $\mathbb{N}$ , Vector A)  $\rightarrow$  Matrix A
- DataBag A  $\rightarrow$  (A  $\rightarrow$  (N, N))  $\rightarrow$  Matrix A



Lara also has an IR with two "views".

A Monadic View that enables Dataflow optimisations → Operator Fusion and Operator Pushdown











Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$

#### **Operations**:

- For-comprehensions
- Linear algebra functions

#### Type conversions track provenance:

- DataBag ( $\mathbb{N}$ , Vector A)  $\rightarrow$  Matrix A
- DataBag A  $\rightarrow$  (A  $\rightarrow$  (N, N))  $\rightarrow$  Matrix A



Lara also has an IR with two "views".

A Monadic View that enables Dataflow optimisations → Operator Fusion and Operator Pushdown











Lara has a Quotation Based DSL embedded in Scala.

Types (Monads):

- DataBag A ::= Multiset A
- Vector A ::= Set (N, A)
- Matrix A ::= Set  $((\mathbb{N}, \mathbb{N}), \mathbb{A})$
- Stream A ::= Infinite Multiset A

#### **Operations**:

- For-comprehensions
- Linear algebra functions

#### Type conversions track provenance:

- DataBag ( $\mathbb{N}$ , Vector A)  $\rightarrow$  Matrix A
- DataBag A  $\rightarrow$  (A  $\rightarrow$  (N, N))  $\rightarrow$  Matrix A



Lara also has an IR with two "views".

A Monadic View that enables Dataflow optimisations → Operator Fusion and Operator Pushdown











## Arc - Compiler Approach







## Arc - Compiler Approach **Original approach Embedded DSL** Arc IR Compiler **Arcon Runtime**







## Arc - Compiler Approach Original approach









# **Original approach**











## What is MLIR (Multi-Level IR)?







### MLIR: A Compiler Infrastructure for the End of **Moore's Law**

Chris Lattner \* Google

Mehdi Amini Google

Uday Bondhugula IISc

Jacques Pienaar Google

**River Riddle** Google

## What is MLIR (Multi-Level IR)?

Albert Cohen Google

Andy Davis Google

Tatiana Shpeisman Google

Nicolas Vasilache Google

#### **Oleksandr Zinenko**

Google











#### Compiler infrastructure of Google's **TensorFlow**







#### Compiler infrastructure of Google's **TensorFlow**



**Problem: No re-use between IR compilers** 





## Compiler infrastructure of Google's **TensorFlow**



#### **Problem: No re-use between IR compilers**

A compiler infrastructure like LLVM's is needed





#### Compiler infrastructure of Google's TensorFlow



#### **Problem: No re-use between IR compilers**

A compiler infrastructure like LLVM's is needed

### Compiler infrastructure of languages using LLVM











#### Compiler infrastructure of Google's **TensorFlow**



#### **Problem: No re-use between IR compilers**

A compiler infrastructure like LLVM's is needed

### Compiler infrastructure of languages using LLVM



**Problem: LLVM is locked to one level of abstraction** 









#### Compiler infrastructure of Google's **TensorFlow**



#### **Problem: No re-use between IR compilers**

→ A compiler infrastructure like LLVM's is needed

#### Compiler infrastructure of languages using LLVM



**Problem: LLVM is locked to one level of abstraction** There is a need for a more general solution













#### MLIR is an **IR** which can be extended with new "dialects"





MLIR is an **IR** which can be extended with new "dialects"

**Dialects define** *operations*, *types*, *type constraints*, *rewrite rules*, *lowerings*, ...





MLIR is an **IR** which can be extended with new "dialects"

**Dialects define** *operations*, *types*, *type constraints*, *rewrite rules*, *lowerings*, ...

Dialects adhere to the same meta-syntax and meta-semantics





MLIR is an **IR** which can be extended with new "dialects"

**Dialects define** operations, types, type constraints, rewrite rules, lowerings, ...

#### **Dialects adhere** to the same **meta-syntax** and meta-semantics

Dialects can coexist in the same program





MLIR is an **IR** which can be extended with new "dialects"

**Dialects define** operations, types, type constraints, rewrite rules, lowerings, ...

#### **Dialects adhere** to the same **meta-syntax** and meta-semantics

Dialects can coexist in the same program

MLIR handles parsing, type checking/ inference, line-number tracking, ...




MLIR is an **IR** which can be extended with new "dialects"

**Dialects define** operations, types, type constraints, rewrite rules, lowerings, ...

#### **Dialects adhere** to the same **meta-syntax** and meta-semantics

Dialects can coexist in the same program

**MLIR handles** *parsing*, *type checking*/ inference, line-number tracking, ...

Additionally, MLIR provides tooling for testing, parallel compilation, documentation, CLI, ...





MLIR is an **IR** which can be extended with new **Builtin dialects (14 total):** "dialects"

**Dialects define** operations, types, type constraints, rewrite rules, lowerings, ...

#### Dialects adhere to the same meta-syntax and meta-semantics

Dialects can coexist in the same program

**MLIR handles** *parsing*, *type checking*/ inference, line-number tracking, ...

Additionally, MLIR provides tooling for testing, parallel compilation, documentation, CLI, ...





MLIR is an **IR** which can be extended with new "dialects"

**Dialects define** operations, types, type constraints, rewrite rules, lowerings, ...

#### Dialects adhere to the same meta-syntax and meta-semantics

Dialects can coexist in the same program

**MLIR handles** *parsing*, *type checking*/ inference, line-number tracking, ...

Additionally, MLIR provides tooling for testing, parallel compilation, documentation, CLI, ...

- **Builtin dialects (14 total):** 
  - Basic types and arithmetics • std





MLIR is an **IR** which can be extended with new **Builtin dialects (14 total):** "dialects" • std

**Dialects define** operations, types, type constraints, rewrite rules, lowerings, ...

#### **Dialects adhere** to the same **meta-syntax** and meta-semantics

Dialects can coexist in the same program

**MLIR handles** *parsing*, *type checking*/ inference, line-number tracking, ...

Additionally, MLIR provides tooling for testing, parallel compilation, documentation, CLI, ...

- Basic types and arithmetics
- Ordinary loops • loop





**Builtin dialects (14 total):** MLIR is an **IR** which can be extended with new "dialects" • std

**Dialects define** operations, types, type constraints, rewrite rules, lowerings, ...

#### **Dialects adhere** to the same **meta-syntax** and meta-semantics

Dialects can coexist in the same program

**MLIR handles** *parsing*, *type checking*/ inference, line-number tracking, ...

Additionally, MLIR provides tooling for testing, parallel compilation, documentation, CLI, ...

- Basic types and arithmetics
- Ordinary loops • loop
- Polyhedral loops • affine





MLIR is an **IR** which can be extended with new **Builtin dialects (14 total):** "dialects" • std

**Dialects define** operations, types, type constraints, rewrite rules, lowerings, ...

#### **Dialects adhere** to the same **meta-syntax** and meta-semantics

Dialects can coexist in the same program

**MLIR handles** *parsing*, *type checking*/ inference, line-number tracking, ...

Additionally, MLIR provides tooling for testing, parallel compilation, documentation, CLI, ...

- Basic types and arithmetics
- Ordinary loops • loop
- Polyhedral loops • affine
- Linear algebra • linalg





MLIR is an **IR** which can be extended with new **Builtin dialects (14 total):** "dialects" • std

**Dialects define** operations, types, type constraints, rewrite rules, lowerings, ...

#### **Dialects adhere** to the same **meta-syntax** and meta-semantics

Dialects can coexist in the same program

MLIR handles parsing, type checking/ inference, line-number tracking, ...

Additionally, MLIR provides tooling for testing, parallel compilation, documentation, CLI, ...

- Basic types and arithmetics
- Ordinary loops • loop
- Polyhedral loops • affine
- linalg - Linear algebra
- CUDA & OpenCL abstraction gpu





MLIR is an **IR** which can be extended with new **Builtin dialects (14 total):** "dialects" • std

**Dialects define** operations, types, type constraints, rewrite rules, lowerings, ...

#### Dialects adhere to the same meta-syntax and meta-semantics

Dialects can coexist in the same program

**MLIR handles** *parsing*, *type checking*/ inference, line-number tracking, ...

Additionally, MLIR provides tooling for testing, parallel compilation, documentation, CLI, ...

- Basic types and arithmetics
- Ordinary loops • loop
- Polyhedral loops • affine
- Linear algebra • linalg
  - CUDA & OpenCL abstraction gpu
- LLVM-IR code generation • llvm





MLIR is an **IR** which can be extended with new **Builtin dialects (14 total):** "dialects" • std

**Dialects define** operations, types, type constraints, rewrite rules, lowerings, ...

#### Dialects adhere to the same meta-syntax and meta-semantics

Dialects can coexist in the same program

**MLIR handles** *parsing*, *type checking*/ inference, line-number tracking, ...

Additionally, MLIR provides tooling for testing, parallel compilation, documentation, CLI, ...

- Basic types and arithmetics
- Ordinary loops • loop
- Polyhedral loops • affine
- Linear algebra • linalg
  - CUDA & OpenCL abstraction gpu
  - LLVM-IR code generation llvm











Meta-syntax (with some parts optional):





- **Meta-syntax** (with some parts optional): • ASSIGNMENT ::= %VAR\_NAME = OPERATION





**Meta-syntax** (with some parts optional):

- ASSIGNMENT ::= %VAR\_NAME = OPERATION





**Meta-syntax** (with some parts optional): • ASSIGNMENT ::= %VAR\_NAME = OPERATION

- OPERATION ::= "DIALECT.OP\_NAME"(ARGS) .....





**Meta-syntax** (with some parts optional):

- ASSIGNMENT ::= %VAR\_NAME = OPERATION

• OPERATION ::= "DIALECT.OP\_NAME"(ARGS) (REGIONS) .....





**Meta-syntax** (with some parts optional):

- ASSIGNMENT ::= %VAR\_NAME = OPERATION
- OPERATION ::= "DIALECT.OP\_NAME"(ARGS) (REGIONS) {ATTRS} .....





**Meta-syntax** (with some parts optional):

- ASSIGNMENT ::= %VAR\_NAME = OPERATION

• OPERATION ::= "DIALECT.OP\_NAME"(ARGS) (REGIONS) {ATTRS} : (TYPES) → (TYPES)





**Meta-syntax** (with some parts optional):

- ASSIGNMENT ::= %VAR\_NAME = OPERATION
- TYPE := !DIALECT.TYPE\_NAME<TYPES>

• OPERATION ::= "DIALECT.OP\_NAME"(ARGS) (REGIONS) {ATTRS} : (TYPES) → (TYPES)





**Meta-syntax** (with some parts optional):

- ASSIGNMENT ::= %VAR\_NAME = OPERATION
- OPERATION := "DIALECT.OP\_NAME"(ARGS) (REGIONS) {ATTRS} : (TYPES) → (TYPES) • TYPE := !DIALECT.TYPE\_NAME<TYPES>

**Meta-semantics:** All values are **SSA**, **typed**, **scoped** (and so on)





**Meta-syntax** (with some parts optional):

- ASSIGNMENT ::= %VAR\_NAME = OPERATION
- ::= !DIALECT.TYPE\_NAME<TYPES>
- OPERATION := "DIALECT.OP\_NAME"(ARGS) (REGIONS) {ATTRS} : (TYPES) → (TYPES) • TYPE

**Meta-semantics:** All values are **SSA**, **typed**, **scoped** (and so on)

#### **Example:**

Compute the max of two values





**Meta-syntax** (with some parts optional):

- ASSIGNMENT ::= %VAR\_NAME = OPERATION
- ::= !DIALECT.TYPE\_NAME<TYPES>
- OPERATION := "DIALECT.OP\_NAME"(ARGS) (REGIONS) {ATTRS} : (TYPES) → (TYPES) • TYPE

**Meta-semantics:** All values are **SSA**, **typed**, **scoped** (and so on)

#### **Example:**

```
Compute the
max of two values
```

```
c = a > b
max = if c {
  a
  else {
  b
```





**Meta-syntax** (with some parts optional):

- ASSIGNMENT ::= %VAR\_NAME = OPERATION
- ::= !DIALECT.TYPE\_NAME<TYPES>
- OPERATION := "DIALECT.OP\_NAME"(ARGS) (REGIONS) {ATTRS} : (TYPES) → (TYPES) • TYPE

**Meta-semantics:** All values are **SSA**, **typed**, **scoped** (and so on)

#### **Example:**

```
Compute the
max of two values
   c = a > b
   max = if c {
      a
     else {
      b
```









**Meta-syntax** (with some parts optional):

- ASSIGNMENT ::= %VAR\_NAME = OPERATION
- ::= !DIALECT.TYPE\_NAME<TYPES>
- OPERATION := "DIALECT.OP\_NAME"(ARGS) (REGIONS) {ATTRS} : (TYPES) → (TYPES) • TYPE

**Meta-semantics:** All values are **SSA**, **typed**, **scoped** (and so on)

#### **Example:**

Compute the max of two values **c** = **a** > **b** max = if c { a else {

b

%c = "arc.greater\_than"(%a, %b)









**Meta-syntax** (with some parts optional):

- ASSIGNMENT ::= %VAR\_NAME = OPERATION
- ::= !DIALECT.TYPE\_NAME<TYPES> • TYPE

**Meta-semantics:** All values are **SSA**, **typed**, **scoped** (and so on)

#### **Example:**

Compute the max of two values **c** = **a** > **b** max = if c { a else {

b

• OPERATION ::= "DIALECT.OP\_NAME"(ARGS) (REGIONS) {ATTRS} : (TYPES) → (TYPES)

%c = "arc.greater\_than"(%a, %b) : (!arc.int, !arc.int) → (!arc.bool)









**Meta-syntax** (with some parts optional):

- ASSIGNMENT ::= %VAR\_NAME = OPERATION
- ::= !DIALECT.TYPE\_NAME<TYPES> • TYPE

**Meta-semantics:** All values are **SSA**, **typed**, **scoped** (and so on)

#### **Example:**

Compute the max of two values **c** = **a** > **b** max = if c { a else { b

%max = "arc.if"(%c)

• OPERATION ::= "DIALECT.OP\_NAME"(ARGS) (REGIONS) {ATTRS} : (TYPES) → (TYPES)

%c = "arc.greater\_than"(%a, %b) : (!arc.int, !arc.int)  $\rightarrow$  (!arc.bool)









**Meta-syntax** (with some parts optional):

- ASSIGNMENT ::= %VAR\_NAME = OPERATION
- ::= !DIALECT.TYPE\_NAME<TYPES> • TYPE

**Meta-semantics:** All values are **SSA**, **typed**, **scoped** (and so on)

#### **Example:**

Compute the max of two values c = a > b max = if c { a else { b

```
%max = "arc.if"(%c) ({
    \bullet \bullet \bullet
}, {
\}) : (arc.bool) \rightarrow (!arc.int)
```

• OPERATION ::= "DIALECT.OP\_NAME"(ARGS) (REGIONS) {ATTRS} : (TYPES) → (TYPES)

%c = "arc.greater\_than"(%a, %b) : (!arc.int, !arc.int) → (!arc.bool)









**Meta-syntax** (with some parts optional):

- ASSIGNMENT ::= %VAR\_NAME = OPERATION
- := !DIALECT.TYPE\_NAME<TYPES> • TYPE

**Meta-semantics:** All values are **SSA**, **typed**, **scoped** (and so on)

#### **Example:**

Compute the max of two values c = a > b max = if c { a else { b

```
%max = "arc.if"(%c) ({
  "arc.yield"(%a)
}, {
  "arc.yield"(%b)
\}) : (arc.bool) \rightarrow (!arc.int)
```

• OPERATION := "DIALECT.OP\_NAME"(ARGS) (REGIONS) {ATTRS} : (TYPES) → (TYPES)

%c = "arc.greater\_than"(%a, %b) : (!arc.int, !arc.int)  $\rightarrow$  (!arc.bool)









**Meta-syntax** (with some parts optional):

- ASSIGNMENT ::= %VAR\_NAME = OPERATION
- OPERATION := "DIALECT.OP\_NAME"(ARGS) (REGIONS) {ATTRS} : (TYPES) → (TYPES) ::= !DIALECT.TYPE\_NAME<TYPES> • TYPE

**Meta-semantics:** All values are **SSA**, **typed**, **scoped** (and so on)

### **Example:**

Compute the max of two values c = a > b max = if c { a } else { b

```
%max = "arc.if"(%c) ({
}, {
\}) : (arc.bool) \rightarrow (!arc.int)
```

- %c = "arc.greater\_than"(%a, %b) : (!arc.int, !arc.int) → (!arc.bool)
  - $"arc.yield"(%a) : (!arc.int) \rightarrow (!arc.int)$
  - $"arc.yield"(\%b) : (!arc.int) \rightarrow (!arc.int)$















The **structure** of each operation is defined using an high-level **DSL**, while **details** are implemented in **C++** 







The **structure** of each operation is defined using an high-level **DSL**, while **details** are implemented in **C++** 

**Example:** 







The **structure** of each operation is defined using an high-level **DSL**, while **details** are implemented in **C++** 

**Example:** 







The **structure** of each operation is defined using an high-level **DSL**, while **details** are implemented in **C++** 

**Example:** 

def GreaterThanOp : Op<"greater\_than", [.....]> {

}







The **structure** of each operation is defined using an high-level **DSL**, while **details** are implemented in **C++** 

**Example:** 







The **structure** of each operation is defined using an high-level **DSL**, while **details** are implemented in C++

**Example:** 

let **summary** = "greater than operation";

def GreaterThanOp : Op<"greater\_than", [.....]> { let description = [{ Returns true if \$left is greater than \$right }]; let arguments = (ins AnyType:\$left, AnyType:\$right);







The **structure** of each operation is defined using an high-level **DSL**, while **details** are implemented in C++

**Example:** 

let summary = "greater than operation"; let **results** 

def GreaterThanOp : Op<"greater\_than", [.....]> { let **description** = [{ Returns true if **\$left** is greater than **\$right** }]; let arguments = (ins AnyType:\$left, AnyType:\$right); = (outs BoolType:\$output);






# **MLIR - Defining Operations**

The **structure** of each operation is defined using an high-level **DSL**, while **details** are implemented in C++

**Example:** 

let summary = "greater than operation"; let **results** 

def GreaterThanOp : Op<"greater\_than", [ArgsAreSameType, NoSideEffect]> { let **description** = [{ Returns true if **\$left** is greater than **\$right** }]; let arguments = (ins AnyType:\$left, AnyType:\$right); = (outs BoolType:\$output);







# **MLIR - Defining Operations**

The **structure** of each operation is defined using an high-level **DSL**, while **details** are implemented in **C++** 

**Example:** 

<pre>let summary = "great let description = [{ Ret let arguments = (ins / let results = (outs // Optional let regions = let verifier = let parser = let printer =</pre>	de	ef <b>G</b>	ceaterThanOp	•	0p<"gr
<pre>let description = [{ Ref let arguments = (ins / let results = (outs // Optional let regions = let verifier = let parser = let printer = }</pre>		let	summary	=	"great
<pre>let arguments = (ins / let results = (outs // Optional let regions = let verifier = let parser = let printer = }</pre>		let	description	=	[{ Ret
<pre>let results = (outs // Optional let regions = let verifier = let parser = let printer = }</pre>		let	arguments	=	(ins A
<pre>// Optional let regions = let verifier = let parser = let printer = }</pre>		let	results	=	(outs
<pre>let regions = let verifier = let parser = let printer = }</pre>		// (	Optional		
<pre>let verifier = let parser = let printer = }</pre>		let	regions	=	•••
<pre>let parser = let printer = }</pre>		let	verifier	=	•••
<pre>let printer = }</pre>		let	parser	=	•••
}		let	printer	=	• • •
	}				

reater\_than", [ArgsAreSameType, NoSideEffect]> {
 er than operation";
 urns true if **\$left** is greater than **\$right** }];
 AnyType:**\$left**, AnyType:**\$right**);
 BoolType:**\$output**);















### **Current approach**







### **Current approach**







### **Current approach**







### **Current approach**







# Arc is a dialect in MLIR for data analytics that takes inspiration from Lara

### **Current approach**







**Arc** is a dialect in **MLIR** for data analytics that takes inspiration from **Lara** 

Arc aims to **extend Lara**'s model with support for **Stream** data types

### **Current approach**







Arc is a dialect in MLIR for data analytics that takes inspiration from Lara

Arc aims to **extend Lara**'s model with support for **Stream** data types

Through MLIR, Arc can **reuse** existing compiler technology and widen its **scope** 

### **Current approach**







Arc is a dialect in MLIR for data analytics that takes inspiration from Lara

Arc aims to **extend Lara**'s model with support for **Stream** data types

Through MLIR, Arc can **reuse** existing compiler technology and widen its **scope** 

Upcoming work: Embedded DSL Design

### **Current** approach







### Extra slides

Presenter: Klas Segeljakt <<u>klasseg@kth.se</u>>



#### KTH VETENSKAP OCH KONST

### Lara's DSL - Example





#### Description

Based on **numerical** and **categorical** data, train a **Ridge Regression** model with **3-fold cross-validation**, and **Mean Squared Error** as the loss function, to the predict number of future clicks on advertisements





#### Description

	1
Based on numerical and categorical	2
data train a <b>Ridge Regression</b> model	3
with 2 fold areas validation and	4
with <b>3-foid cross-validation</b> , and	5
Mean Squared Error as the loss function,	6
to the predict number of future clicks on	7
advertisements	8
	9
	10
	11
	12
	13
	14
	15
	10
	10
	10
	20
	20
	22
	23
	24
	25
	26
	27





Description	In	ρι
	1 /	1
Based on <b>numerical</b> and <b>categorical</b>	2 /	/
data train a <b>Ridge Regression</b> model	3 V	al
	4	
with <b>3-told cross-validation</b> , and	5	
Mean Squared Error as the loss function,	6	
to the predict number of future clicks on	7	
advartisaments	8	
auvertiserrierits	9	
	10	
	11	
	12	
	13	
	14	
	15	
	16	
	17	
	18	
	19	
	20	
	21	
	22	
	23	
	24	
	25	
	26	
	27	

#### put data: [L,N,N,N,N,N,N,N,N,N,N,C,C,C,C]

/ Column 0 contains the target variable, columns 1-10 contain / numerical and columns 11-15 contain categorical features al dataset = readAndClean("/path/to/data")





## Lara's DSL - Example

Description	Inpu
	1 // 0
Based on numerical and categorical	2 // n
data train a <b>Ridge Regression</b> model	3 <b>val</b>
with 2 fold areas validation and	4 <b>val</b>
with <b>3-ioid cross-validation</b> , and	5 <b>val</b>
Mean Squared Error as the loss function,	6 <b>val</b>
to the predict number of future clicks on	7
advertisements	8
	9
	10
	11
	12
	13
	14
	16
	17
	18
	19
	20
	21
	22
	23
	24
	25
	26
	27

#### it data: [L,N,N,N,N,N,N,N,N,N,N,C,C,C,C]

Column 0 contains the target variable, columns 1-10 contain numerical and columns 11-15 contain categorical features

- dataset = readAndClean("/path/to/data")
- encoded = dummyEncode(dataset, 11 to 15)
- vectors = concatNumericalFeatures(encoded, 1 to 10)
- features = concatVectors(vectors)





## Lara's DSL - Example

Description	I	npu
Description Based on numerical and categorical data, train a Ridge Regression model with 3-fold cross-validation, and Mean Squared Error as the loss function, to the predict number of future clicks on advertisements	$     1 \\     2 \\     3 \\     4 \\     5 \\     6 \\     7 \\     8 \\     9 \\     10 \\     11 \\     12 \\     13 \\     14 \\     15 \\     16 \\     17 \\     18 \\     19 \\     20 \\     21 \\     22 \\     23 \\     24 \\     25 \\     26 \\     $	npu // ( // i val val val val val val val
	27	

#### it data: [L,N,N,N,N,N,N,N,N,N,C,C,C,C]

Column 0 contains the target variable, columns 1-10 contain numerical and columns 11-15 contain categorical features dataset = readAndClean("/path/to/data") encoded = dummyEncode(dataset, 11 to 15) vectors = concatNumericalFeatures(encoded, 1 to 10) features = concatVectors(vectors) y = 0: extract 1st column as target vector y = Matrix(features, y = 0) (M, y) = Matrix.normalize(M, 1 to 10) Х





Description	I	npu
	1	// 0
Based on <b>numerical</b> and <b>categorical</b> data train a <b>Bidge Begression</b> model	2 3	// n val
with <b>3-fold cross-validation</b> and	4	val
Many Courses of Freeze as the loss function	5	val
mean Squared Error as the loss function,	6	var // w
to the predict number of future clicks on	( 8	// y val
advertisements	9	val
	10	// 0
	11	val
	12	for
	13	
	14	
	15	
	16	
	17	
	18	
	19	
	20	
	21	
	22	
	23	
	25	
	26	
	27	}

#### out data: [L,N,N,N,N,N,N,N,N,N,N,C,C,C,C,C]

```
/ Column 0 contains the target variable, columns 1-10 contain
/ numerical and columns 11-15 contain categorical features
al dataset = readAndClean("/path/to/data")
al encoded = dummyEncode(dataset, 11 to 15)
al vectors = concatNumericalFeatures(encoded, 1 to 10)
al features = concatVectors(vectors)
/ y = 0: extract 1st column as target vector y
al (M, y) = Matrix(features, y = 0)
al X = Matrix.normalize(M, 1 to 10)
/ Grid search over hyperparameter candidates
al regCandidates: Seq[Double] = // ...
or (lambda <- regCandidates) {</pre>
```





Description	I	npu
	1	// 0
Based on numerical and categorical	2	// n
data, train a <b>Ridge Regression</b> model	3	val
with <b>3-fold cross-validation</b> and	4	val
	5	val
mean Squared Error as the loss function,	6	val
to the predict number of future clicks on	7	// y
advertisements	8	val
	10	// 0
	11	val
	12	for
	13	11
	14	va
	15	
	16	
	17	
	18	
	19	
	20	
	21	
	22	
	23	
	24	}
	25	//
	26	pr ר
	27	3

#### out data: [L,N,N,N,N,N,N,N,N,N,N,C,C,C,C]

```
Column 0 contains the target variable, columns 1-10 contain
numerical and columns 11-15 contain categorical features
dataset = readAndClean("/path/to/data")
encoded = dummyEncode(dataset, 11 to 15)
vectors = concatNumericalFeatures(encoded, 1 to 10)
features = concatVectors(vectors)
y = 0: extract 1st column as target vector y
(M, y) = Matrix(features, y = 0)
         = Matrix.normalize(M, 1 to 10)
Х
Grid search over hyperparameter candidates
regCandidates: Seq[Double] = // ...
(lambda <- regCandidates) {
/ 3-fold cross-validation for the hyperparameter lambda
al errors = ML.crossValidate(3, X, y) {
(X_train, X_test, y_train, y_test) =>
```

}
// Print mean error for chosen hyperparameter
println(errors.sum / k)





Description	I	npu
	1	// 0
Based on numerical and categorical	2	// n
data, train a <b>Ridge Regression</b> model	3	val
with <b>3-fold cross-validation</b> and	4	val
	5	val
mean Squared Error as the loss function,	6	vai
to the predict number of future clicks on	7	// y
advertisements	8	val
	10	// 0
	11	val
	12	for
	13	11
	14	va
	15	
	16	
	17	
	18	
	19	
	20	
	21	
	22	
	23	
	24	}
	25	//
	26	pr ר
	27	3

#### out data: [L,N,N,N,N,N,N,N,N,N,N,C,C,C,C]

```
Column 0 contains the target variable, columns 1-10 contain
numerical and columns 11-15 contain categorical features
dataset = readAndClean("/path/to/data")
encoded = dummyEncode(dataset, 11 to 15)
vectors = concatNumericalFeatures(encoded, 1 to 10)
features = concatVectors(vectors)
y = 0: extract 1st column as target vector y
(M, y) = Matrix(features, y = 0)
         = Matrix.normalize(M, 1 to 10)
Х
Grid search over hyperparameter candidates
regCandidates: Seq[Double] = // ...
(lambda <- regCandidates) {
/ 3-fold cross-validation for the hyperparameter lambda
al errors = ML.crossValidate(3, X, y) {
(X_train, X_test, y_train, y_test) =>
// Ridge regression
```

}
// Print mean error for chosen hyperparameter
println(errors.sum / k)





Description	Ι	np
	1	//
Based on numerical and categorical	2	//
data, train a Ridge Regression model	3	va.
with <b>3-fold cross-validation</b> , and	4	va. va
Mean Squared Frror as the loss function	6	va
to the predict number of future clicks on	7	//
advartia amonto	8	va
adventisements	9	va
	10	//
	11	va
	12	Í0:
	13	
	14	
	16	
	17	
	18	
	19	
	20	
	21	
	22	
	23	
	24	
	20	1

27 }

#### out data: [L,N,N,N,N,N,N,N,N,N,N,C,C,C,C]

```
Column 0 contains the target variable, columns 1-10 contain
numerical and columns 11-15 contain categorical features
l dataset = readAndClean("/path/to/data")
1 encoded = dummyEncode(dataset, 11 to 15)
l vectors = concatNumericalFeatures(encoded, 1 to 10)
1 features = concatVectors(vectors)
y = 0: extract 1st column as target vector y
1 (M, y) = Matrix(features, y = 0)
          = Matrix.normalize(M, 1 to 10)
lΧ
Grid search over hyperparameter candidates
l regCandidates: Seq[Double] = // ...
r (lambda <- regCandidates) {
// 3-fold cross-validation for the hyperparameter lambda
val errors = ML.crossValidate(3, X, y) {
 (X_train, X_test, y_train, y_test) =>
 // Ridge regression
```

}
// Print mean error for chosen hyperparameter
println(errors.sum / k)





Description	Ι	'n
	1	1
Based on numerical and categorical	2	1
data, train a <b>Ridge Regression</b> model	3	v
with <b>3-fold cross-validation</b> and	4	v
Mean Squared Error on the loss function	5	V
iviean Squared Error as the loss function,	0 7	
to the predict number of future clicks on	8	v
advertisements	9	v
	10	1
	11	v
N K	12	f
$\hat{\mathbf{w}} = \operatorname{argmin}  \sum_{n=1}^{N} (\mathbf{w} - \mathbf{v}  \mathbf{w})^2 + \lambda \sum_{n=1}^{N} \mathbf{w}^2$	13	
$\mathbf{w}_{ridge} - \arg \min_{\mathbf{w}} \sum (\mathbf{y}_i - \mathbf{A}_i \mathbf{w}) + \mathbf{A} \sum \mathbf{w}_i$	14	
i=1 $i=1$	15	
	10	
	18	
	19	
	20	
	21	
	22	
	23	
	24	
	25	
	26	

27 }

#### put data: [L,N,N,N,N,N,N,N,N,N,C,C,C,C]

```
/ Column 0 contains the target variable, columns 1-10 contain
/ numerical and columns 11-15 contain categorical features
al dataset = readAndClean("/path/to/data")
al encoded = dummyEncode(dataset, 11 to 15)
al vectors = concatNumericalFeatures(encoded, 1 to 10)
al features = concatVectors(vectors)
/ y = 0: extract 1st column as target vector y
al (M, y) = Matrix(features, y = 0)
           = Matrix.normalize(M, 1 to 10)
al X
/ Grid search over hyperparameter candidates
al regCandidates: Seq[Double] = // ...
or (lambda <- regCandidates) {</pre>
// 3-fold cross-validation for the hyperparameter lambda
val errors = ML.crossValidate(3, X, y) {
  (X_train, X_test, y_train, y_test) =>
  // Ridge regression
```

// Print mean error for chosen hyperparameter println(errors.sum / k)





Description	In	
	1 /	/
Based on numerical and categorical	2 /	/
data, train a <b>Ridge Regression</b> model	3 1	v
with <b>3-fold cross-validation</b> . and	4	v v
Mean Squared Frror as the loss function	6 1	v
to the prodict number of future clicks on	7	/
advertigensente	8	v
adventisements	9 1	V
	10 /	/
	11	V
N K	12 1	f
$\hat{\mathbf{w}}_{ii} = \arg \min \sum (\mathbf{v}_i - \mathbf{X}_i \mathbf{w})^2 + \lambda \sum \mathbf{w}_i^2$	13	
$\frac{W}{W} = \frac{1}{1} \left( \frac{J}{V} - \frac{J}{V} \right) + \frac{J}{V} = \frac{1}{1} \left( \frac{J}{V} - \frac{J}{V} \right)$	15	
$l=1 \qquad l=1$	16	
$\hat{\mathbf{W}}_{ridge} = (\mathbf{X}^{T}\mathbf{X} + \lambda\mathbf{I})^{-1}(\mathbf{X}^{T}\mathbf{y})$	17	
	18	
	19	
	20	
	21	
	22	
	23	
	24	

26

27 }

#### put data: [L,N,N,N,N,N,N,N,N,N,N,C,C,C,C]

```
/ Column 0 contains the target variable, columns 1-10 contain
/ numerical and columns 11-15 contain categorical features
al dataset = readAndClean("/path/to/data")
al encoded = dummyEncode(dataset, 11 to 15)
al vectors = concatNumericalFeatures(encoded, 1 to 10)
al features = concatVectors(vectors)
/ y = 0: extract 1st column as target vector y
al (M, y) = Matrix(features, y = 0)
           = Matrix.normalize(M, 1 to 10)
al X
// Grid search over hyperparameter candidates
al regCandidates: Seq[Double] = // ...
or (lambda <- regCandidates) {</pre>
// 3-fold cross-validation for the hyperparameter lambda
val errors = ML.crossValidate(3, X, y) {
  (X_train, X_test, y_train, y_test) =>
  // Ridge regression
```

}
// Print mean error for chosen hyperparameter
println(errors.sum / k)





Description	Ι	'n
Based on <b>numerical</b> and <b>categorical</b> data, train a <b>Ridge Regression</b> model with <b>3-fold cross-validation</b> , and <b>Mean Squared Error</b> as the loss function, to the predict number of future clicks on advertisements	1 2 3 4 5 6 7 8 9	// v v v v v / v
	10 11	v / v
$\hat{\mathbf{w}}_{ridge} = \arg\min_{\mathbf{w}} \sum_{i=1}^{N} \left( \mathbf{y}_{i} - \mathbf{X}_{i} \mathbf{w} \right)^{2} + \lambda \sum_{i=1}^{K} \mathbf{w}_{i}^{2}$ $\hat{\mathbf{w}}_{ridge} = (\mathbf{X}^{\mathrm{T}} \mathbf{X} + \lambda \mathbf{I})^{-1} (\mathbf{X}^{\mathrm{T}} \mathbf{y})$ $\hat{\mathbf{w}}_{ridge} = (\mathbf{X}^{\mathrm{T}} \mathbf{X} + \lambda \mathbf{I}) \setminus (\mathbf{X}^{\mathrm{T}} \mathbf{y})$	12 13 14 15 16 17 18	T
"ridge = (11 11 + 701) + (11 3)	19 20 21 22 23 24	

27 }

26

#### put data: [L,N,N,N,N,N,N,N,N,N,N,C,C,C,C]

```
/ Column 0 contains the target variable, columns 1-10 contain
/ numerical and columns 11-15 contain categorical features
ral dataset = readAndClean("/path/to/data")
al encoded = dummyEncode(dataset, 11 to 15)
al vectors = concatNumericalFeatures(encoded, 1 to 10)
al features = concatVectors(vectors)
// y = 0: extract 1st column as target vector y
ral (M, y) = Matrix(features, y = 0)
           = Matrix.normalize(M, 1 to 10)
al X
// Grid search over hyperparameter candidates
al regCandidates: Seq[Double] = // ...
or (lambda <- regCandidates) {
// 3-fold cross-validation for the hyperparameter lambda
val errors = ML.crossValidate(3, X, y) {
   (X_train, X_test, y_train, y_test) =>
  // Ridge regression
```

}
// Print mean error for chosen hyperparameter
println(errors.sum / k)





Description	Ι	'n
	1	/
Based on <b>numerical</b> and <b>categorical</b>	2	/
data train a <b>Ridge Regression</b> model	3	V
with 2 fold areas validation and	4	V
with <b>3-told cross-validation</b> , and	5	V
Mean Squared Error as the loss function,	6	V
to the predict number of future clicks on	7	/
advertisements	8	V T
	10	/
	11	v
$\lambda I$ V	12	f
$\mathbf{x} = \mathbf{x} + $	13	
$\mathbf{w}_{ridge} = \arg\min \sum (\mathbf{y}_i - \mathbf{X}_i \mathbf{w}) + \lambda \sum \mathbf{w}_i^2$	14	
i=1 $i=1$	15	
$\hat{\mathbf{W}}_{ridae} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda \mathbf{I})^{-1}(\mathbf{X}^{\mathrm{T}}\mathbf{y})$	16 17	
$\mathbf{A} = (\mathbf{x}_{T}\mathbf{x}_{T} \cdot \mathbf{A}\mathbf{x}_{T} \cdot \mathbf{A}x$	18	
$\mathbf{W}_{ridge} = (\mathbf{X}^{T}\mathbf{X} + \lambda\mathbf{I}) \setminus (\mathbf{X}^{T}\mathbf{y}) \boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{$	19	
	20	
	$^{21}$	
	22	
	23	
	24	
	25	
	26	

27 }

#### put data: [L,N,N,N,N,N,N,N,N,N,N,C,C,C,C]

```
// Column 0 contains the target variable, columns 1-10 contain
// numerical and columns 11-15 contain categorical features
val dataset = readAndClean("/path/to/data")
ral encoded = dummyEncode(dataset, 11 to 15)
val vectors = concatNumericalFeatures(encoded, 1 to 10)
val features = concatVectors(vectors)
// y = 0: extract 1st column as target vector y
val (M, y) = Matrix(features, y = 0)
            = Matrix.normalize(M, 1 to 10)
val X
// Grid search over hyperparameter candidates
regCandidates: Seq[Double] = // ...
or (lambda <- regCandidates) {
 // 3-fold cross-validation for the hyperparameter lambda
 val errors = ML.crossValidate(3, X, y) {
   (X_train, X_test, y_train, y_test) =>
   // Ridge regression
```

}
// Print mean error for chosen hyperparameter
println(errors.sum / k)





Description	Ι	'n
	1	/
Based on <b>numerical</b> and <b>categorical</b>	2	/
data train a <b>Ridge Regression</b> model	3	V
with 2 fold areas validation and	4	V
with <b>3-told cross-validation</b> , and	5	V
Mean Squared Error as the loss function,	6	V
to the predict number of future clicks on	7	/
advertisements	8	V T
	10	/
	11	v
$\lambda I$ V	12	f
$\mathbf{x} = \mathbf{x} + $	13	
$\mathbf{w}_{ridge} = \arg\min \sum (\mathbf{y}_i - \mathbf{X}_i \mathbf{w}) + \lambda \sum \mathbf{w}_i^2$	14	
i=1 $i=1$	15	
$\hat{\mathbf{W}}_{ridae} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda \mathbf{I})^{-1}(\mathbf{X}^{\mathrm{T}}\mathbf{y})$	16 17	
$\mathbf{A} = (\mathbf{x}_{T}\mathbf{x}_{T} \cdot \mathbf{A}\mathbf{x}_{T} \cdot \mathbf{A}x$	18	
$\mathbf{W}_{ridge} = (\mathbf{X}^{T}\mathbf{X} + \lambda\mathbf{I}) \setminus (\mathbf{X}^{T}\mathbf{y}) \boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{$	19	
	20	
	$^{21}$	
	22	
	23	
	24	
	25	
	26	

27 }

#### put data: [L,N,N,N,N,N,N,N,N,N,N,C,C,C,C]

```
// Column 0 contains the target variable, columns 1-10 contain
// numerical and columns 11-15 contain categorical features
val dataset = readAndClean("/path/to/data")
ral encoded = dummyEncode(dataset, 11 to 15)
val vectors = concatNumericalFeatures(encoded, 1 to 10)
val features = concatVectors(vectors)
// y = 0: extract 1st column as target vector y
val (M, y) = Matrix(features, y = 0)
            = Matrix.normalize(M, 1 to 10)
val X
// Grid search over hyperparameter candidates
regCandidates: Seq[Double] = // ...
or (lambda <- regCandidates) {
 // 3-fold cross-validation for the hyperparameter lambda
 val errors = ML.crossValidate(3, X, y) {
   (X_train, X_test, y_train, y_test) =>
   // Ridge regression
   val reg = Matrix.eye(X_train.nCols) * lambda
   val XtX = X_train.t ** X_train + reg
   val Xty = X_train.t ** y_train
   val w = XtX \setminus Xty
```

}
// Print mean error for chosen hyperparameter
println(errors.sum / k)





Description	Ι	'n
	1	/
Based on <b>numerical</b> and <b>categorical</b>	2	/
data train a <b>Ridge Regression</b> model	3	V
with 2 fold areas validation and	4	V
with <b>3-told cross-validation</b> , and	5	V
Mean Squared Error as the loss function,	6	V
to the predict number of future clicks on	7	/
advertisements	8	V T
	10	/
	11	v
$\lambda I$ V	12	f
$\mathbf{x} = \mathbf{x} + $	13	
$\mathbf{w}_{ridge} = \arg\min \sum (\mathbf{y}_i - \mathbf{X}_i \mathbf{w}) + \lambda \sum \mathbf{w}_i^2$	14	
i=1 $i=1$	15	
$\hat{\mathbf{W}}_{ridae} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda \mathbf{I})^{-1}(\mathbf{X}^{\mathrm{T}}\mathbf{y})$	16 17	
$\mathbf{A} = (\mathbf{x}_{T}\mathbf{x}_{T} \cdot \mathbf{A}\mathbf{x}_{T} \cdot \mathbf{A}x$	18	
$\mathbf{W}_{ridge} = (\mathbf{X}^{T}\mathbf{X} + \lambda\mathbf{I}) \setminus (\mathbf{X}^{T}\mathbf{y}) \boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{\boldsymbol{$	19	
	20	
	$^{21}$	
	22	
	23	
	24	
	25	
	26	

27 }

#### put data: [L,N,N,N,N,N,N,N,N,N,N,C,C,C,C]

```
// Column 0 contains the target variable, columns 1-10 contain
// numerical and columns 11-15 contain categorical features
val dataset = readAndClean("/path/to/data")
ral encoded = dummyEncode(dataset, 11 to 15)
val vectors = concatNumericalFeatures(encoded, 1 to 10)
val features = concatVectors(vectors)
// y = 0: extract 1st column as target vector y
val (M, y) = Matrix(features, y = 0)
            = Matrix.normalize(M, 1 to 10)
val X
// Grid search over hyperparameter candidates
regCandidates: Seq[Double] = // ...
or (lambda <- regCandidates) {
 // 3-fold cross-validation for the hyperparameter lambda
 val errors = ML.crossValidate(3, X, y) {
   (X_train, X_test, y_train, y_test) =>
   // Ridge regression
   val reg = Matrix.eye(X_train.nCols) * lambda
   val XtX = X_train.t ** X_train + reg
   val Xty = X_train.t ** y_train
   val w = XtX \setminus Xty
   // Calculate mean squared error on test set
```

// Print mean error for chosen hyperparameter
println(errors.sum / k)





Description	Ι	r
	1	/
Based on numerical and categorical	2	1
data, train a Ridge Regression model	3	1
with <b>3-fold cross-validation</b> , and	5	1
Mean Squared Error as the loss function.	6	1
to the predict number of future clicks on	$\overline{7}$	
advartisamente	8	
auventisementis	9	
	10	1
	11	
$\frac{N}{2}$	12	
$\hat{\mathbf{w}}_{ridge} = \arg\min \left[ \sum_{i}^{\prime} (\mathbf{y}_{i} - \mathbf{X}_{i}\mathbf{w})^{2} + \lambda \right] \mathbf{w}_{i}^{2}$	14	
$\mathbf{W}  \mathbf{W}  $	15	
$\hat{\mathbf{w}} = (\mathbf{v} \mathbf{T} \mathbf{v} + 2\mathbf{I}) - 1(\mathbf{v} \mathbf{T}_{\mathbf{v}})$	16	
$\mathbf{w}_{ridge} = (\mathbf{\Lambda} \ \mathbf{\Lambda} + \mathbf{\Lambda}\mathbf{I}) \ (\mathbf{\Lambda} \ \mathbf{y})$	17	
$\hat{\mathbf{W}}_{ridge} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda \mathbf{I}) \setminus (\mathbf{X}^{\mathrm{T}}\mathbf{y})$	18	
	19	
	20 21	
$1 \sum_{N=1}^{N} (1 \sum_$	21 22	
$\mathbf{WSE} = -\frac{1}{N}\sum_{i} (\mathbf{y}_{i} - \mathbf{X}_{i}\mathbf{W})$	23	
i = 1	<b>24</b>	
	25	

- 26
- 27 }

#### nput data: [L,N,N,N,N,N,N,N,N,N,N,C,C,C,C,C]

```
// Column 0 contains the target variable, columns 1-10 contain
// numerical and columns 11-15 contain categorical features
val dataset = readAndClean("/path/to/data")
val encoded = dummyEncode(dataset, 11 to 15)
val vectors = concatNumericalFeatures(encoded, 1 to 10)
val features = concatVectors(vectors)
// y = 0: extract 1st column as target vector y
val (M, y) = Matrix(features, y = 0)
            = Matrix.normalize(M, 1 to 10)
val X
// Grid search over hyperparameter candidates
val regCandidates: Seq[Double] = // ...
for (lambda <- regCandidates) {</pre>
 // 3-fold cross-validation for the hyperparameter lambda
 val errors = ML.crossValidate(3, X, y) {
   (X_train, X_test, y_train, y_test) =>
   // Ridge regression
   val reg = Matrix.eye(X_train.nCols) * lambda
   val XtX = X_train.t ** X_train + reg
   val Xty = X_train.t ** y_train
   val w = XtX \setminus Xty
   // Calculate mean squared error on test set
```

// Print mean error for chosen hyperparameter
println(errors.sum / k)





Description	Ι	:n
	1	
Based on numerical and categorical	2	/
data. train a <b>Ridge Regression</b> model	3	
with <b>3-fold cross-validation</b> and	4	
Moon Squared Error as the loss function	о 6	,
to the predict purchas affuture alighteres	7	
to the predict number of future clicks on	8	1
advertisements	9	٦
	10	
	11	
N K	12	
$\hat{\mathbf{w}}_{1} = \operatorname{argmin} \sum (\mathbf{v}_{1} - \mathbf{X}_{1}\mathbf{w})^{2} + \lambda \sum \mathbf{w}_{1}^{2}$	13	
$\frac{\text{Wridge}}{W} = \frac{W}{1} \left( \frac{J_i}{J_i} + \frac{J_i}{I_i} \right) + \frac{J_i}{I_i} \left( \frac{J_i}{I_i} + \frac{J_i}{I_i} \right) + \frac{J_i}$	$14 \\ 15$	
l=1 $l=1$ $l=1$	16	
$\hat{\mathbf{w}}_{ridge} = (\mathbf{X}^{T}\mathbf{X} + \lambda\mathbf{I})^{-1}(\mathbf{X}^{T}\mathbf{y})$	17	
$\hat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} \perp \lambda \mathbf{I}) \setminus (\mathbf{X}^{\mathrm{T}}\mathbf{v})$	18	
ridge – (IX IX I / II) ( (IX Y)	19	
	20	
$1 \frac{N}{2}$	21	
$MSE = \frac{1}{2} \sum_{i=1}^{n} \left[ (\mathbf{y}_{i} - \mathbf{X}_{i} \mathbf{w})^{2} \right]$	22	
$N \stackrel{\checkmark}{=} 1 \stackrel{(\bullet i)}{=} 1$	23	
$\iota - 1$	24	

26

27 }

#### nput data: [L,N,N,N,N,N,N,N,N,N,N,C,C,C,C,C]

```
// Column 0 contains the target variable, columns 1-10 contain
// numerical and columns 11-15 contain categorical features
val dataset = readAndClean("/path/to/data")
val encoded = dummyEncode(dataset, 11 to 15)
val vectors = concatNumericalFeatures(encoded, 1 to 10)
val features = concatVectors(vectors)
// y = 0: extract 1st column as target vector y
val (M, y) = Matrix(features, y = 0)
            = Matrix.normalize(M, 1 to 10)
val X
// Grid search over hyperparameter candidates
val regCandidates: Seq[Double] = // ...
for (lambda <- regCandidates) {</pre>
 // 3-fold cross-validation for the hyperparameter lambda
 val errors = ML.crossValidate(3, X, y) {
   (X_train, X_test, y_train, y_test) =>
   // Ridge regression
   val reg = Matrix.eye(X_train.nCols) * lambda
   val XtX = X_train.t ** X_train + reg
   val Xty = X_train.t ** y_train
   val w = XtX \setminus Xty
   // Calculate mean squared error on test set
```

// Print mean error for chosen hyperparameter
println(errors.sum / k)





Description	Ι	:n
	1	
Based on numerical and categorical	2	/
data. train a <b>Ridge Regression</b> model	3	
with <b>3-fold cross-validation</b> and	4	
Moon Squared Error as the loss function	о 6	,
to the predict purchas affuture alighteres	7	
to the predict number of future clicks on	8	1
advertisements	9	٦
	10	/
	11	
N K	12	
$\hat{\mathbf{w}}_{1} = \operatorname{argmin} \sum (\mathbf{v}_{1} - \mathbf{X}_{1}\mathbf{w})^{2} + \lambda \sum \mathbf{w}_{1}^{2}$	13	
$\frac{\text{Wridge}}{W} = \frac{W}{1} \left( \frac{J_i}{J_i} + \frac{J_i}{I_i} \right) + \frac{J_i}{I_i} \left( \frac{J_i}{I_i} + \frac{J_i}{I_i} \right) + \frac{J_i}$	$14 \\ 15$	
l=1 $l=1$ $l=1$	16	
$\hat{\mathbf{w}}_{ridge} = (\mathbf{X}^{T}\mathbf{X} + \lambda\mathbf{I})^{-1}(\mathbf{X}^{T}\mathbf{y})$	17	
$\hat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} \perp \lambda \mathbf{I}) \setminus (\mathbf{X}^{\mathrm{T}}\mathbf{v})$	18	
ridge – (IX IX I / II) ( (IX Y)	19	
	20	
$1 \frac{N}{2}$	21	
$MSE = \frac{1}{2} \sum_{i=1}^{n} \left[ (\mathbf{y}_{i} - \mathbf{X}_{i} \mathbf{w})^{2} \right]$	22	
$N \stackrel{\checkmark}{=} 1 \stackrel{(\bullet i)}{=} 1$	23	
$\iota - 1$	24	

26

27 }

#### nput data: [L,N,N,N,N,N,N,N,N,N,N,C,C,C,C,C]

```
// Column 0 contains the target variable, columns 1-10 contain
// numerical and columns 11-15 contain categorical features
val dataset = readAndClean("/path/to/data")
val encoded = dummyEncode(dataset, 11 to 15)
val vectors = concatNumericalFeatures(encoded, 1 to 10)
val features = concatVectors(vectors)
// y = 0: extract 1st column as target vector y
val (M, y) = Matrix(features, y = 0)
            = Matrix.normalize(M, 1 to 10)
val X
// Grid search over hyperparameter candidates
val regCandidates: Seq[Double] = // ...
for (lambda <- regCandidates) {</pre>
 // 3-fold cross-validation for the hyperparameter lambda
 val errors = ML.crossValidate(3, X, y) {
   (X_train, X_test, y_train, y_test) =>
   // Ridge regression
   val reg = Matrix.eye(X_train.nCols) * lambda
   val XtX = X_train.t ** X_train + reg
   val Xty = X_train.t ** y_train
   val w = XtX \setminus Xty
   // Calculate mean squared error on test set
   val residuals = y_test - (X_test ** w)
   residuals.map(r => r * r).agg(_ + _) / y_test.size
 // Print mean error for chosen hyperparameter
 println(errors.sum / k)
```





### References

[1] Beam.apache.org. (2020). Apache Beam. [online] Available at: https://beam.apache.org/ [Accessed 2 Mar. 2020].

[2] Walt, S.V.D., Colbert, S.C. and Varoquaux, G., 2011. The **NumPy** array: a structure for efficient numerical computation. *Computing in Science & Engineering*, *13*(2), pp.22-30.

[3] Ragan-Kelley, J., Barnes, C., Adams, A., Paris, S., Durand, F. and Amarasinghe, S., 2013. **Halide**: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *Acm Sigplan Notices*, *48*(6), pp.519-530.

[4] Date, C.J. and Darwen, H., 1993. A Guide to the SQL Standard (Vol. 3). Reading: Addison-wesley.

[5] Arasu, A., Babu, S. and Widom, J., 2006. The **CQL** continuous query language: semantic foundations and query execution. *The VLDB Journal*, *15*(2), pp.121-142.

[6] Kunft, A., Katsifodimos, A., Schelter, S., Breß, S., Rabl, T. and Markl, V., 2019. An intermediate representation for optimizing machine learning pipelines. *Proceedings of the VLDB Endowment*, *12*(11), pp.1553-1567.

[7] Malewicz, G., Austern, M.H., Bik, A.J., Dehnert, J.C., Horn, I., Leiser, N. and Czajkowski, G., 2010, June. **Pregel**: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* (pp. 135-146).

[8] Iyer, A.P., Pu, Q., Patel, K., Gonzalez, J.E. and Stoica, I., 2019. *TEGRA: Efficient ad-hoc analytics on time-evolving graphs*. Technical report.

[9] Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A. and Schreiber, R.S., 2013, April. Presto: distributed machine learning and graph processing with sparse matrices. In *Proceedings of the 8th ACM European Conference on Computer Systems* (pp. 197-210).

[10] Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Rydberg, M., Selmer, P. and Taylor, A., 2018, May. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data* (pp. 1433-1445).

