Linfa

Classical Machine Learning with Rust

Lorenz Schmidt [1] Paul Koerbitz [2]

[1] *RWTH Aachen*

[2] *Google*

November 26, 2020

INTRODUCTION

Classical Machine Learning

- Dominant field until 2010, before release of AlexNet

Classical Machine Learning

- Dominant field until 2010, before release of AlexNet

Classical Machine Learning

- Dominant field until 2010, before release of AlexNet
- *PCA, SVM, ElasticNet, kernel methods, …*

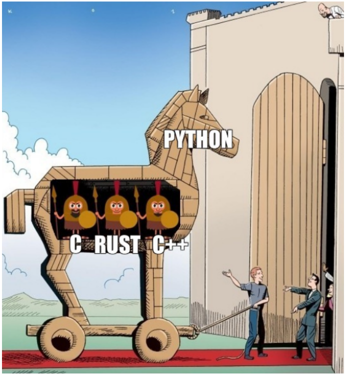
Classical Machine Learning

- Dominant field until 2010, before release of AlexNet
- *PCA*, *SVM*, *ElasticNet*, *kernel methods*, …
- But still easier to interpret and faster if domain suits

Linfa's Goal

Linfa's Goal

- Unified workhorse



Linfa's Goal

- Unified workhorse
- Speed and correctness of Rust (+ Ecosystem)
  - Use borrowing system

```
let dataset = Dataset::new(data, targets);
let (train, valid) = dataset.split_with_ratio(0.9);
```

LINFA'S GOAL

- Unified workhorse
- Speed and correctness of Rust (+ Ecosystem)
  - Use borrowing system
  - No GC, no runtime, deployment for IoT

LINFA'S GOAL

- Unified workhorse
- Speed and correctness of Rust (+ Ecosystem)
  - Use borrowing system
  - No GC, no runtime, deployment for IoT
  - Excellent tools for testing, benchmarking

Linfa's Goal

- Unified workhorse
- Speed and correctness of Rust (+ Ecosystem)
- Offer reference implementations and guide for developers

Linfa's Goal

- Unified workhorse
- Speed and correctness of Rust (+ Ecosystem)
- Offer reference implementations and guide for developers
  - Focus on understandable implementations

LINFA'S GOAL

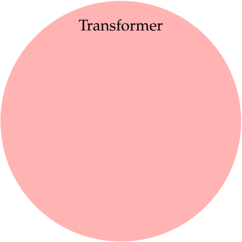
- Unified workhorse
- Speed and correctness of Rust (+ Ecosystem)
- Offer reference implementations and guide for developers
  - Focus on understandable implementations
  - Integrate Rust-Book with suitable LaTeX

Linfa's Goal

- Unified workhorse
- Speed and correctness of Rust (+ Ecosystem)
- Offer reference implementations and guide for developers
- Explore language features for safe algorithm design

Classes of Algorithms

Classes of Algorithms

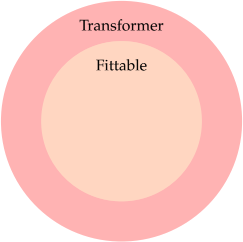


- Examples
  - Kernel methods
  - *PCA*, Diffusion maps
  - t-SNE

```rust
pub trait Transformer<R: Records, T> {
    fn transform(&self, x: R) -> T;
}
```

Classes of Algorithms

- Examples
  - Decision trees
  - Hierarchical clustering
  - DBSCAN, Optics

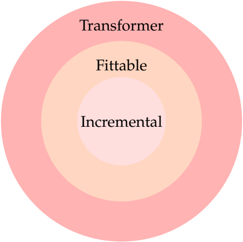



```rust
pub trait Fit<R: Records, T: Targets> {
    type Object;

    fn fit(&self, dataset: &Dataset<R, T>) -> Self::Object;
}
```

Classes of Algorithms

- Examples
  - ElasticNet
  - Gaussian mixture model
  - Fast ICA




```rust
pub trait IncrementalFit<R: Records, T: Targets> {
    type Object;

    fn fit_with<I>(&self, model: I, dataset: &Dataset<R, T>) -> Self::Object
        where I: Into<Option<Self::Object>>;
}
```

Dataset

- Offer unified interface for data and targets

Dataset

- Offer unified interface for data and targets
- Associated types for late binding in *Records*, *Targets* traits

```rust
pub struct Dataset<R, T>
where
    R: Records,
    T: Targets,
{
    records: R,
    targets: T,

    weights: Vec<f32>,
}
```

Dataset

- Offer unified interface for data and targets
- Associated types for late binding in *Records*, *Targets* traits
- Differentiate between probabilities, floating values and discrete labels

```rust
// Probability type
pub struct Pr(f32);

// Floating value, implemented for f32/f64
pub trait Float: NdFloat + Default + .. {}

// Discrete Label, implemented for int/str/bool
pub trait Label: Eq + Hash + Clone {}
```

Metric - Classification

Metric - Classification

- Use confusion matrix as entry point to various classification metrics

```
let cm = pred.confusion_matrix(&ground_truth);
println!("accuracy {:.2}, precision {:.2}, MCC {:.2}",
  cm.accuracy(), cm.precision(), cm.mcc());
```

Metric - Classification

- Use confusion matrix as entry point to various classification metrics
- ROC curves for binary classifications

```
// prediction is probability, ground_truth binary label
let roc = pred.roc(&ground_truth);
let curve = roc.get_curve();
let auc = roc.area_under_curve();
```

Dataset - Examples

- K-Folding

```
Dataset::new(data, targets)
    .fold(12)
    .into_iter()
    .map(|(train, valid)| {
        let model = params.fit(&train);
        let predi = model.predict(&valid);
        predi.confusion_matrix(&valid)
    })
    .map(|cm| cm.accuracy())
    .sum() / 12.0;
```

Dataset - Examples

- K-Folding
- One-vs-All

```
dataset.one_vs_all()
  .map(|(x, label)| (params.fit(&x), label))
  .collect::<Vec<_>>()
  .into_multi_model();
```

K-Means

- First available algorithm, implemented at RustFest 2019

K-Means

- First available algorithm, implemented at RustFest 2019
- Simple Expectation-Maximization optimization with hard assignment

K-Means

- First available algorithm, implemented at RustFest 2019
- Simple Expectation-Maximization optimization with hard assignment
- Done in two days, performance already improved when compared to *Scikit-learn*

Library	Training time
Linfa	476.2 ms
Scikit-learn	604.7 ms

Table: Mean training time for model of 1 mio data points

DIFFUSION MAPS

- Connectivity as transition probability matrix **M**

$$\mathcal{P}(X_{t+1} = j | X_t = i) = M_{ij} \tag{1}$$

Diffusion maps

- Connectivity as transition probability matrix **M**

$$\mathcal{P}(X_{t+1} = j | X_t = i) = M_{ij} \quad (1)$$

- Where are we after t steps, starting from node v_i

$$v_i \to \mathbf{e}_i^T \mathbf{M}^t = (M_{i1}^{(t)}, \ldots, M_{in}^{(t)}) \quad (2)$$

Diffusion maps

- Connectivity as transition probability matrix **M**

$$\mathcal{P}(X_{t+1} = j | X_t = i) = M_{ij} \quad (1)$$

- Where are we after t steps, starting from node v_i

$$v_i \rightarrow \mathbf{e}_i^T \mathbf{M}^t = (M_{i1}^{(t)}, \ldots, M_{in}^{(t)}) \quad (2)$$

- Perform spectral decomposition of **M** and project onto axis along it diffuses

Diffusion maps - Example

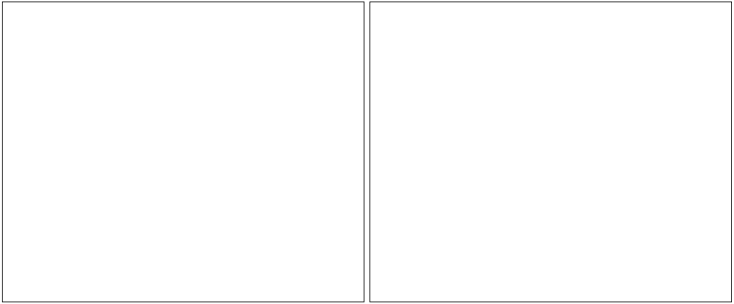
```rust
// generate sparse RBF kernel with eps=2.0
let kernel = Kernel::params()
    .kind(KernelType::Sparse(15))
    .method(KernelMethod::Gaussian(2.0))
    .transform(&dataset);

// embed similarity matrix with diffusion maps
let embedding = DiffusionMap::<f64>::params(2)
    .steps(1)
    .build()
    .transform(&kernel);

// get embedding
let embedding = embedding.embedding();
```

Figure: Kernel method and diffusion maps invocation in Linfa

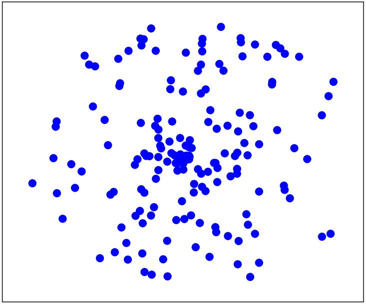
DIFFUSION MAPS - EXAMPLE



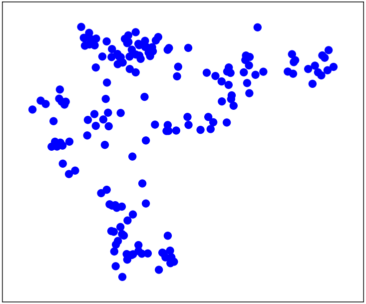
(a) Original (b) Embedded

Figure: Nested rings and its transformation with diffusion maps.

Diffusion maps - Example



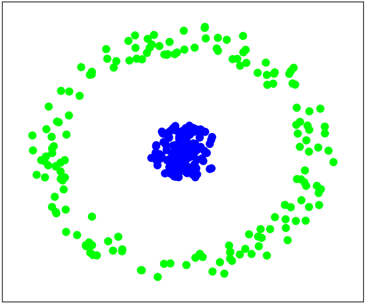
(a) Original



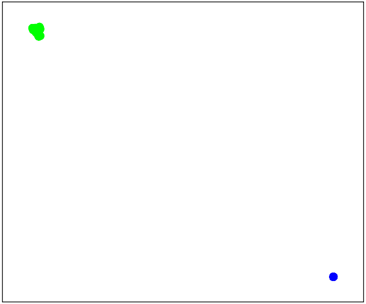
(b) Embedded

Figure: Nested rings and its transformation with diffusion maps.

Diffusion maps - Example



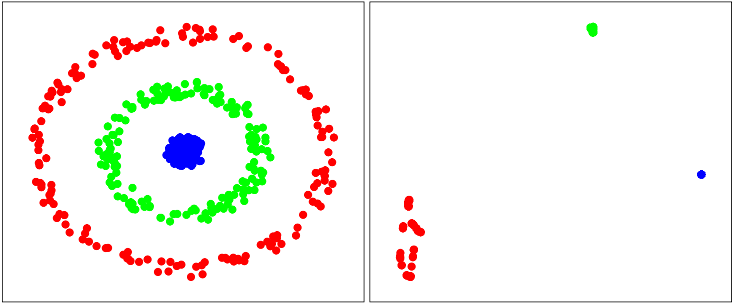
(a) Original



(b) Embedded

Figure: Nested rings and its transformation with diffusion maps.

Diffusion maps - Example



(a) Original (b) Embedded

Figure: Nested rings and its transformation with diffusion maps.

Diffusion maps - LOBPCG

Diffusion maps - LOBPCG

- Locally Optimal Block Preconditioned Conjugate

Conversation 11 · Commits 30 · Checks 7 · Files changed 14

bytesnake commented on Mar 12 • edited · Contributor

This PR ports the LOBPCG algorithm from scipy to Rust. The algorithm is useful for the symmetric eigenproblem for just a couple of eigenvalues (for example for multidimensional scaling of gaussian kernels). Solves the issue #160

I did not implement the generalized eigenproblem for matrix `B` different to identity, as its a uncommon use-case (at least in machine-learning), but if required the modification should be minor.

It also adds access to the functions `ssygv`, `dsygv`, `zhegv`, `chegv` for the generalized eigenvalue problem

$$\mathbf{A}\phi_i = \lambda_i \mathbf{B}\phi_i, \qquad (i = 1, \cdots, n)$$

with additional mass matrix `B`. The traits are implemented for tuples of `A` and `B`, so you can use it like this

```
let (eigvals, (eigvecs, B_cholesky)) = (A, B).eigh(UPLO::Upper);
```

Remaining issues:

- [x] Implement the orthogonalization to the constraint matrix `Y`
- [x] Improve documentation of the `lobpcg.rs` file and add examples
- [x] Implement truncated eigenvalue decomposition
- [x] Implement truncated SVD based on this
- [x] Benchmark the implementation
- [x] Add restart routine if cholesky fails

Diffusion maps - LOBPCG

- Locally Optimal Block Preconditioned Conjugate
  - useful for finding largest eigenvalues and corresponding eigenvectors
  - factorization free, i.e. useful for sparse matrices

Diffusion maps - LOBPCG

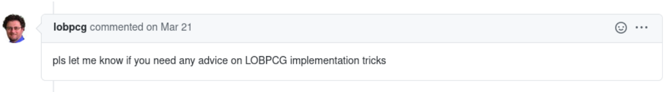
- Locally Optimal Block Preconditioned Conjugate
  - useful for finding largest eigenvalues and corresponding eigenvectors
  - factorization free, i.e. useful for sparse matrices
  - matrix free, i.e. useful for covariance matrices

Diffusion maps - LOBPCG

- Locally Optimal Block Preconditioned Conjugate
  - useful for finding largest eigenvalues and corresponding eigenvectors
  - factorization free, i.e. useful for sparse matrices
  - matrix free, i.e. useful for covariance matrices
  - linear convergence theoretically guaranteed

DIFFUSION MAPS - LOBPCG

- Locally Optimal Block Preconditioned Conjugate
  - useful for finding largest eigenvalues and corresponding eigenvectors
  - factorization free, i.e. useful for sparse matrices
  - matrix free, i.e. useful for covariance matrices
  - linear convergence theoretically guaranteed



Available Algorithms

- Clustering: K-Means, Gaussian Mixture, DBSCAN, *Optics*

Available Algorithms

- Clustering: K-Means, Gaussian Mixture, DBSCAN, *Optics*
- Support Vector Machines: C/ν/one-class classification and ϵ/ν regression

Available Algorithms

- Clustering: K-Means, Gaussian Mixture, DBSCAN, *Optics*
- Support Vector Machines: C/ν/one-class classification and ϵ/ν regression
- Reduction: PCA, diffusion maps and kernel methods

AVAILABLE ALGORITHMS

- Clustering: K-Means, Gaussian Mixture, DBSCAN, *Optics*
- Support Vector Machines: C/ν/one-class classification and ϵ/ν regression
- Reduction: PCA, diffusion maps and kernel methods
- Logistic regression

Available Algorithms

- Clustering: K-Means, Gaussian Mixture, DBSCAN, *Optics*
- Support Vector Machines: C/ν/one-class classification and ϵ/ν regression
- Reduction: PCA, diffusion maps and kernel methods
- Logistic regression
- Decision trees

AVAILABLE ALGORITHMS

- Clustering: K-Means, Gaussian Mixture, DBSCAN, *Optics*
- Support Vector Machines: C/ν/one-class classification and ϵ/ν regression
- Reduction: PCA, diffusion maps and kernel methods
- Logistic regression
- Decision trees
- Hierarchical clustering

Available Algorithms

- Clustering: K-Means, Gaussian Mixture, DBSCAN, *Optics*
- Support Vector Machines: C/ν/one-class classification and ϵ/ν regression
- Reduction: PCA, diffusion maps and kernel methods
- Logistic regression
- Decision trees
- Hierarchical clustering
- Ordinary least squares, generalize linear models, *elastic net*

Available Algorithms

- Clustering: K-Means, Gaussian Mixture, DBSCAN, *Optics*
- Support Vector Machines: C/ν/one-class classification and ϵ/ν regression
- Reduction: PCA, diffusion maps and kernel methods
- Logistic regression
- Decision trees
- Hierarchical clustering
- Ordinary least squares, generalize linear models, *elastic net*
- Fast Independent Component Analysis

Available Algorithms

- Clustering: K-Means, Gaussian Mixture, DBSCAN, *Optics*
- Support Vector Machines: C/ν/one-class classification and ϵ/ν regression
- Reduction: PCA, diffusion maps and kernel methods
- Logistic regression
- Decision trees
- Hierarchical clustering
- Ordinary least squares, generalize linear models, *elastic net*
- Fast Independent Component Analysis
- *Gaussian Naive Bayes*

Available Algorithms

- Clustering: K-Means, Gaussian Mixture, DBSCAN, *Optics*
- Support Vector Machines: C/ν/one-class classification and ϵ/ν regression
- Reduction: PCA, diffusion maps and kernel methods
- Logistic regression
- Decision trees
- Hierarchical clustering
- Ordinary least squares, generalize linear models, *elastic net*
- Fast Independent Component Analysis
- *Gaussian Naive Bayes*
- *Ensemble algorithms*, *Random Forest*

NEXT STEPS

Next Steps

- Good progress in Roadmap

Roadmap #7

Open LukeMathWalker opened this issue on Dec 2, 2019 · 44 comments

LukeMathWalker commented on Dec 2, 2019 • edited by bytesnake Member

In terms of functionality, the mid-term end goal is to achieve an offering of ML algorithms and pre-processing routines comparable to what is currently available in Python's `scikit-learn`.

These algorithms can either be:

- re-implemented in Rust;
- re-exported from an existing Rust crate, if available on crates.io with a compatible interface.

In no particular order, focusing on the main gaps:

- Clustering:
  - [x] DBSCAN
  - [x] Spectral clustering;
  - [x] Hierarchical clustering;
  - [ ] OPTICS.
- Preprocessing:
  - [x] PCA
  - [x] ICA
  - [ ] Normalisation (**@InCogNiTo124** is working on it)
  - [ ] CountVectoriser (**@bplevin36** is working on it)
  - [ ] TFIDF (**@bplevin36** is working on it)
- Supervised Learning:
  - [x] Linear regression;
  - [ ] Ridge regression; (**@paulkoerbitz**)
  - [ ] LASSO; (**@paulkoerbitz**)
  - [ ] ElasticNet; (**@paulkoerbitz**)
  - [x] Support vector machines;
  - [ ] Nearest Neighbours; (**@mstallmo** is working on it)
  - [ ] Gaussian processes; (integrating `friedrich` - tracking issue nestordemeure/friedrich#1)
  - [x] Decision trees;
  - [ ] Random Forest (**@fgadaleta**)

NEXT STEPS

- Good progress in Roadmap
- But: proper testing is hard and still needs improvement

Next Steps

- Good progress in Roadmap
- But: proper testing is hard and still needs improvement
- Currently unifying and polishing interfaces

Next Steps

- Good progress in Roadmap
- But: proper testing is hard and still needs improvement
- Currently unifying and polishing interfaces
- Improve documentation in some crates

Next Steps

- Good progress in Roadmap
- But: proper testing is hard and still needs improvement
- Currently unifying and polishing interfaces
- Improve documentation in some crates
- And most important of all: do some real-world experiments

Thank you For Your Attention

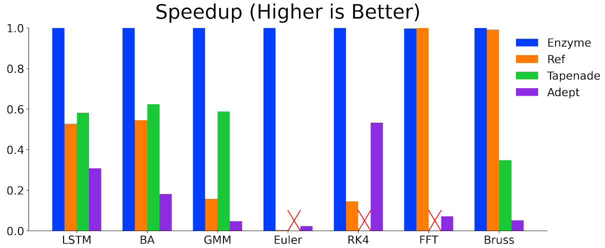
Thank you For Your Attention

And To All Contributors

WHAT IS ABOUT AD AND GPU?

What is about AD and GPU?

- Enzyme: Automatically generate differentiated functions in LLVM



What is about AD and GPU?

- Enzyme: Automatically generate differentiated functions in LLVM
- Highly parallel execution on GPUs