Proposal for GSoC 2020

Improving the user experience

About Me

BASIC INFORMATION

Name: Yuhui Liu
Email: liuyuhui002@foxmail.com
GitHub Account: LiuYuHui

EDUCATION

Pattern Recognition & Intelligent Systems, expected graduation July 2022

PULL REQUESTS

The following PRs have been merged :

- **#4918**, Add feature importance for C45
- **#4899**, Add feature importances for RandomForest
- **#4887**, Fix CARTree leaks
- **#4869**, Add feature importances for CARTree
- **#4837**, Use if constexpr in DenseDistance
- **#4806**, Add feature Levenshtein distance
- **#4834**, Add MeanSquared unittest
- **#4885**, Port libsvmoneclass example
- **#4831**, Use linage in KMeansMiniBatch
- **#4962**, Change KMeans "mus" parameter

Project

Project Name

Improving the user experience

Project Description

GSoC 2020 project usability

Outline

1. **Refactor the base class Machine API**

   Shogun now has a very different APIs compared to other ML libraries, it may be confusing for the people from other libraries, so it's important to redesign the current user API. The `fit` and `predict` is used by most ML libraries, such as sklearn, so we can add those two methods to the base class Machine.

   ```
   Machine::fit(shared_ptr<Features>data, shared_ptr<Labels> lab);
   Machine::fit(shared_ptr<Features> data);
   template<typename T,
               std::enable_if_t<std::is_base_of_v<Labels, T>, int> = 0>
   Machine::predict(shared_ptr<Features> data);
   ```

   and then we can call those two methods like this

```
// method 1
auto classifier = Machine()
        .fit(X,y).predict<MulticlassLabels>(X2);
// method 2
auto classifiler = Machine();
Machine.fit(X.y);
Machine.predict<MulticlassLabels>(X2);
```

Since shogun already have train_machine in every Machine class, we can simply implement fit method like this

```
Machine::fit(shard_ptr<Features> data, shared_ptr<labels> labs){
       set_labels(labs);
       return train_machine(datas);
}
```

2. Composite : Pipeline

Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. Since shogun already has the ensemble class BaggingMachine, we can combine multiple algorithms together. But the class is only used by Random Forest. Therefore, adding a class Composite can do those combinations is essential.

```
// just a simple implementation
class Composite : std::enable_shared_from_this<Composite> {
public:
       Composite() = default;
       Composite(std::vector<Transformer>&& trans ) :m_trans(std::move(trans)){}
       ~Composite() = default;
       std::shared_ptr<Composite> with(shared_ptr<Machine> machine){
           m_bags.push_back(machine);
           return shared_from_this();
       }
       std::shared_ptr<Composite> then(shared_ptr<CombinationRule> rule){
           m_rule = rule;
           return shared_from_this();
       }
       std::shared_ptr<BaggingMachine>
       fit(shared_ptr<Features> datas,shared_ptr<Labels> y){
               auto curr_data = datas;
               for(auto && tran: m_trans){
                     curr_data = tran.transforms(curr_data);
               }
            return make_shared<BaggingMachine>(rule, m_bags)->fit(curr_data, y);
       }
private:
       std::vector<shared_ptr<Transformer>> m_trans;
       std::vector<shared_ptr<Machine>> m_bags;
       std::shared_ptr<CombinationRule> m_rule;
};
```

then we can call use Composite as below.

```
auto clf = composite()
                    -> with(make_shared<LibSVM>())
                    -> with(make_shared<NearestNeighbor>())
                    -> then(make_shared<MeanRule>())
clf->fit(X,y);
```

And we can make this cooler, since Wuwei Lin have already added Transformer and Pipeline to Shogun, we can combine Pipeline with Composite together.

```
shared_ptr<Composite> Pipeline::composite(){
       return make_shared<Composite>(m_stages);
}
auto clf = Pipeline()
                     -> over(transformer)
                     -> composite()
                          -> with(make_shared<LibSVM>())
                          -> with(make_shared<NearestNeighbor>())
                          -> then(make_shared<MeanRule>())
```

Currently, transformer have to combine with machine, it's convenient to have a meta transformer, it can be implemented like this

```
void PipelineBuilder::transform(std::shared_ptr<Features> features>)
{
    auto curr_data = features;
    for (auto stage : stages){
           auto transformer = stage->as<Transformer>();
           transformer.transform(curr_data);
     }
}
// we use the thansform like this
auto meta_trans = std::make_shared<PipelineBuilder>()
                           ->over(transformer1)
                           ->over(transformer2)
                            ->transform(data);
```

3. **Connect custom shogun exception to the SWIG interfaces**

   It is desirable to catch exception that occur in C++ functions and propagate them up to the scripting language interface. By default, SWIG does nothing, but we can create a user-definable exception handler using the %exception directive.

   Since some custom exception have been added to the shogun, it is essential to connect those exceptions to the SWIG interfaces. The exception.i library of SWIG provides support for creating language independent exceptions in interfaces. So, we can implement this connection like this:

```
%exception {
       try {
              $action
       } catch(NotFittedException) {
```

```
                SWIG_exception(SWIG_SystemError, "a machine or a transformer
has not been fitted");
        } catch(InvalidStateException) {
                SWIG_exception(SWIG_SystemError, "an object in Shogun in
invalid state");
        } catch(ShogunNotImplementedException) {
                SWIG_exception(SWIG_SystemError, "this method has not been
implemented");
        } catch(...) {
                SWIG_exception(SWIG_RuntimeError,"Unknown exception");
        }
}
```

4. **Add cross-validation wrapper for the Machine class**

   It is nice to add the cross-validation wrapper to the shogun. The basis idea is add a cv class, and the fit method in the cv class do the cross-validation.

```
template<typename MachineType>
class CV : public MachineType
{
public:
        CV(std::initializer_list<float64_t> parms) : MachineType(),
m_parms(parms) {}
        void fit(std::shared_ptr<Feature> Feature) override{
        //do the cross-validation
        }
private:
        std::initializer_list<float64_t> m_parms;
};
// use the cv class like this
auto clf = CV<Ridge>{0.1, 1.0, 10.0)}.fit(data);
```

5. **Suggest class name if class name not found**

   When a wrong class name to be used in factory, a exception that "Class {} with primitive type {} does not exist." is threw. It's maybe very confusing when people don't know the correct class name, so it's great to suggest the correct class name to users. The basic idea is add a custom WrongNameException in C++ side, and then use SWIG to propagate this exception to other languages.

```
template <class T>
std::shared_ptr<T> create_object(const char* name,
    EPrimitiveType pt = PT_NOT_GENERIC) noexcept(false)
{
    auto object = create(name, pt);
    if (!object)
    {
          //error(
            //"Class {} with primitive type {} does not exist.", name,
            //ptype_name(pt).c_str());
           auto correct_name = find_correct_name(name);
           error<WrongNameException>("Class {} does    m,
             not exist. Did you mean {}" ,name, correct_name);
     }
    auto cast = std::dynamic_pointer_cast<T>(object);
    if (cast == nullptr)
    {
        error("could not cst");
    }
    return cast;
}
```

Expected Timeline

Date	Work
Community Bonding	• Discuss with the mentors to elaborate the design. • Keep diving into shogun's code.
Week 1 - 3	• Refactor Machine APIs and implement fit and predict for some derived class • Add some cookbooks and meta example for the new fit and predict
Week 4	• Connect custom shogun exception to the SWIG interfaces
Week 5	• Suggest class names if class name not found #4473 • Suggest class name in factory instantiation #4475
Milestone First Evaluation	• By this time, we will have new Machine API fit and predict
Week 6 - 8	• Implement the Composite class to chain multiple machines • Write cookbooks and meta example for new Composite stuff
Milestone Second Evaluation	• By this time, we have cool composite stuff, users can easily combine multiple machine learning algorithms.
Week 9	• Combine Pipeline with Composite together • **Write a notebook showcasing the fit/predict stuff in combination with the pipelines**

Week 10	• Implement the help to expose parameter documentation, such as help(SVM.c)
Week 11	• Implement the cross-validation wrapper
Week 12	• Clean up shogun code with modern C++, improve code readability • Finish documents and get prepared for final evaluation.

Extra Information

Skills (Rate 1 - 5)

- **Modern C++ (4)**
  Spent many times learning C++, including reading some C++ books (C++ Primer, Effective C++, Effective Modern C++). Besides, CppCon also teaches me lots about writing good C++ programs.
- **C (3)**
  Know some pattern in C, and use C to implement a tiny operator system (ucore)
- **Python (2)**
  Know basis syntax, but not very familiar with.
- **CMake (2)**
  Know basis usage, but not very familiar with.
- **SWIG (2)**
  Know basis usage, but not very familiar with.
- **Machine Learning (2)**
  Learn some ML courses on Coursera and know basic ML knowledge.

Working Time

I will be based in China during the summer. Therefore, I will be working in UTC +8 timezone. I can work at least 40 hours per week.

Reason for Participation

I enjoy writing code, especially writing elegant c++ code. The shogun need improving a lot, I am happy to contribute my code to making the shogun cooler. I got involved with shogun in January this year and have been implementing some cool stuff including feature importance, levenshtein distance etc. **I will stay around after GSoC**. I am looking forward to working on the project with shogun!