Titanic: Machine Learning vs GP vs EMADE

Austin Peng, Nikhil Vangala, David Zhang, Leul Wubete
Automated Algorithm Design VIP

10/25/2021

Data Preprocessing

- Dropped PassengerID, Ticket, and Cabin
- Added a column for “relatives” which is the sum of sibsp (siblings/spouses aboard) and parch (parents/children aboard)
- **What we changed: Used one-hot encoding for genders as well as name titles.**
- Mapped age ranges and fare ranges to integers
- Added a column “gender_embarked” that takes into account both gender and embarked and set to 0, 1 based on likelihood of survival from plotting gender and embarked port
  - embarked = Q, gender = F => 1
  - embarked = C, gender F => 0
- Replaced missing values with the mean of the feature

Machine Learning

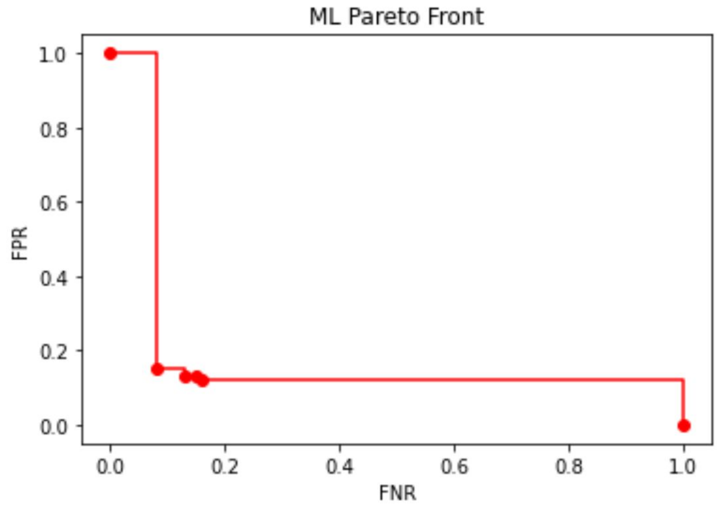
Machine Learning Models Tested

- We started with a variety of machine learning models popular for classification problems including SVM, Random Forests, KNN, Logistic Regression, MLP.
- After testing general performance of each algorithm, we decided that SVM, Random Forests, Logistic Regression, and MLP were the most effective models
- From the pair of models we each tried to optimize the model and tweak hyperparameters offered within sklearn but were unable to create a non-dominated pareto front
- We removed SVM and used the Gaussian Naive Bayes classifier since it was able to produce a model with a higher discrepancy between false negatives and false positives (which made it easier to create the non-dominated pareto front)

Non-Dominated Pareto Front (Machine Learning)

Model	# False Negatives	# False Positives
Gaussian Naive Bayes	40	21
MLP	30	29
Random Forest	30	30
Logistic Regression	31	25

Non-Dominated Pareto Front (Machine Learning)




```
Area Under Curve: 0.19219999999999998
```

Multi Objective Genetic Programminng

Genetic Programming

- We used strongly typed primitives to enforce boolean outcome
- Params
  - Selection: tools.selLexicase (randomly choose order of objectives to compare individuals)
  - Mate: one point crossover
  - Mutate: mutUniform
- Primitives
  - Mainly added logic primitives (and, or, not...etc) and some arithmetic primitives (add, subtract, multiply...etc)
- Evaluation function returns tuple of FN^2, FP^2 which are the objectives we are trying to minimize
  - **What we changed: add third objective of tree size to evaluation function instead of appending a penalizing term to the original objectives.**
  - Adding the additional penalizing factor increased speed of algorithm considerably
  - If false negatives or false positives are over a certain threshold, we penalize heavily (FN >= positives or FP >= negatives)
- Set max tree height for mate and mutation to 17

GP Evaluation Function

```
def evaluation_func_multi(individual, x_train, y_train, pset):
    func = gp.compile(expr=individual, pset=pset)
    predictions = func(x_train[cols[0]],x_train[cols[1]],x_train[cols[2]],x_train[cols[3]],x_train[cols[4]],x_train[col
    confusion = confusion_matrix(y_train, predictions)
    FN = confusion[1,0]
    FP = confusion[0,1]
    positives = np.sum(confusion, axis=1)[0]
    negatives = np.sum(confusion, axis=1)[1]

    if FN >= positives or FP > negatives:
        return (1000000, 1000000)
    e1 = FN**2 + len(individual) * 20
    e2 = FP**2 + len(individual) * 20
    return (e1, e2)
```

Machine Learning vs GP

- Built in sklearn methods made it very easy to optimize and find a solution
- GP constrained by primitives and terminals, whereas ML algorithms are constrained by architecture
- Tree size limit for GP (<91) makes limits complexity of algorithm compared to ML algorithms like MLP
- Both ML and GP algorithms dependent on loss functions
- ML results seemed to be more consistent between runs (GP mating and mutation resulted in highly different final individuals and GP sometimes gets “stuck” or shows increasing loss over time)

EMADE

EMADE: setup

- Used a combination of pip and conda installs to download all dependencies
- initialized **new** conda env with version python 3.7, **not** changing base env to 3.7
- Fixed program compiling failures through changing gtMOEP.py to the latest code commit

```diff
 class MyPool(multiprocess.pool.Pool):
-    Process = NoDaemonProcess
+    @staticmethod
+    def Process(ctx, *args, **kwds):
+        return NoDaemonProcess(*args, **kwds)
```

EMADE: setup continued

- added FNR/FPR to our input file, originally only false negatives and false positives
- Modified sel_nsga2 to achieve selTournamentDCD()'s requirement of having an "individuals" array divisible by 4.

```xml
<objectives>
    <objective>
        <name>False Positives Rate</name>
        <weight>-1.0</weight>
        <achievable>1</achievable>
        <goal>0</goal>
        <evaluationFunction>false_positive_rate</evaluationFunction>
        <lower>0</lower>
        <upper>1</upper>
    </objective>
    <objective>
        <name>False Negatives Rate</name>
        <weight>-1.0</weight>
        <achievable>1</achievable>
        <goal>0</goal>
        <evaluationFunction>false_negative_rate</evaluationFunction>
        <lower>0</lower>
        <upper>1</upper>
    </objective>
```

```python
def sel_nsga2(individuals, k):
    # NSGA2 algorithm first calls for an assigning of crowding distances handled by selNSGA2
    # This has actually already been done in the main loop
    #sorted_pop = tools.selNSGA2(individuals, k)
    # Next use a binary tournament with ties broken by crowding distance to select the pop
    remain = 4 - len(individuals) % 4
    to_add = []
    while remain:
        to_add.append(individuals[remain])
        remain -= 1
    individuals += to_add
    selected_pop = tools.selTournamentDCD(individuals, k)
    return selected_pop
```

EMADE: getting MySQL to work

- For master process: created a new MySQL user
    - 'guest'@'%', where the '%' character represents a wildcard character in MySQL which in this case is used as an expression to match all possible hostnames and IPs.
- Edit the input file xml to the current guest credentials, hostname changed to master process's IP address followed by their port. Allowed reuse to save progress.
- VITAL STEP: allow port forwarding on the master's home router if the workers will be joining remotely. No VPN was needed to connect after this change.

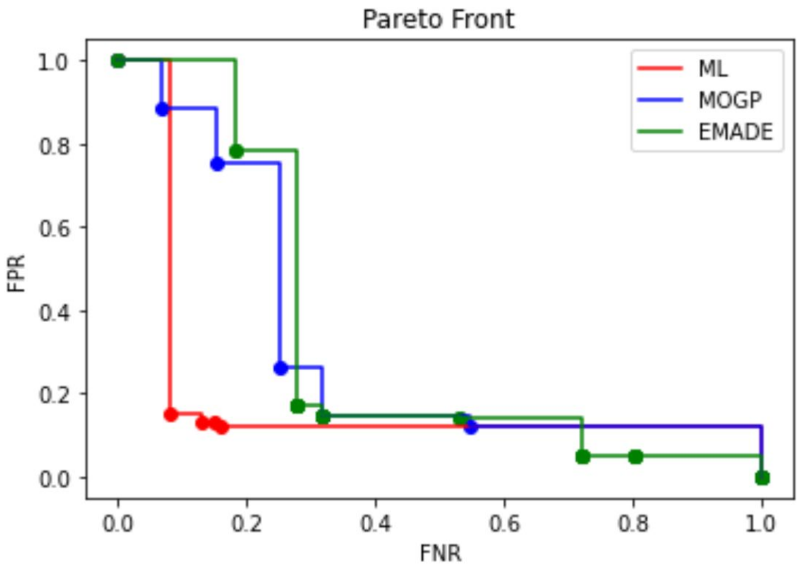
Evaluation Functions

- Edited eval functions to evaluate false positive rate and false negative rate

```
def false_positive_rate(individual, test_data, truth_data, name=None):
    test_data = np.array([elem[0] for elem in test_data])
    truth_data = np.array(truth_data)
    if truth_data.shape != test_data.shape:
        return np.inf
    FP = np.sum(test_data[truth_data==0] != 0)
    TN = np.sum(test_data[truth_data==0] == 0)
    return 0 if FP + TN == 0 else FP/(FP + TN)


def false_negative_rate(individual, test_data, truth_data, name=None):
    test_data = np.array([elem[0] for elem in test_data])
    truth_data = np.array(truth_data)
    if truth_data.shape != test_data.shape:
        return np.inf
    FN = np.sum(test_data[truth_data==1] != 1)
    TP = np.sum(test_data[truth_data==1] == 1)
    return 0 if FN + TP == 0 else FN/(FN + TP)
```

Pareto Front Comparisons




```
Area Under Curve ML: 0.19219999999999998
Area Under Curve MOGP: 0.3222412404349577
Area Under Curve EMADE: 0.33686808080930003
```

Which trees gave most individuals on pareto graph?

- SELECT tree, count(*) as 'total' FROM titanic.individuals natural join titanic.paretofront group by tree order by count(*) DESC;

mostCommonTrees

tree	total
myPlanckTaper(AdaBoostLearner(myGaussian(ARG0, 100.0, 1), ModifyLearnerInt(learnerType('Bayes', None), falseBool, 5), passTriState(1), myIntToFloat(9)), passFloat(myFloatAdd(100.0, 0.1)), passTriState(passT	8
AdaBoostLearner(ARG0, learnerType('LogR', {'penalty': 0, 'C': 1.0}), 2, 0.01)	5
AdaBoostLearner(myProd(ARG0, 0), ModifyLearnerFloat(learnerType('LogR', {'penalty': 0, 'C': 1.0}), 0.01), myIntAdd(3, falseBool), myFloatDiv(0.1, 0.1))	5
BaggedLearner(ARG0, learnerType('Blup', None))	5
AdaBoostLearner(ARG0, learnerType('Bayes', None), 10, myFloatSub(0.01, -2.7166859581983815))	5
AdaBoostLearner(ARG0, learnerType('Bayes', None), 2, 0.01)	4
AdaBoostLearner(ARG0, learnerType('RandForest', {'n_estimators': 100, 'class_weight': 0, 'criterion': 0}), 10, 0.01)	4
AdaBoostLearner(myProd(ARG0, 1), ModifyLearnerFloat(learnerType('Trees', {'criterion': 0, 'splitter': 0}), 0.01), myIntAdd(3, lessThanOrEqual(1.0, 100.0)), myFloatDiv(1.0, 0.1))	4
AdaBoostLearner(ARG0, learnerType('Boosting', {'learning_rate': 0.1, 'n_estimators': 100, 'max_depth': 3}), 2, 0.01)	4
AdaBoostLearner(ARG0, learnerType('SVM', {'C': 1.0, 'kernel': 0}), 10, 0.01)	4
BaggedLearner(ARG0, learnerType('ExtraTrees', {'n_estimators': 100, 'max_depth': 6, 'criterion': 0}))	3
AdaBoostLearner(myProd(ARG0, 2), ModifyLearnerFloat(learnerType('LogR', {'penalty': 0, 'C': 1.0}), 0.01), myIntAdd(3, falseBool), myFloatDiv(1.0, 0.1))	3
AdaBoostLearner(ARG0, learnerType('ExtraTrees', {'n_estimators': 100, 'max_depth': 6, 'criterion': 0}), 10, 0.01)	3
GridSearchLearner(mySelFdr(ARG0, lessThanOrEqual(0.01, 100.0), passFloat(100.0)), ModifyLearnerFloat(ModifyLearnerBool(learnerType('Boosting', {'learning_rate': 0.1, 'n_estimators': 100, 'max_depth': 3}), true	3
AdaBoostLearner(myProd(ARG0, 2), ModifyLearnerFloat(learnerType('LogR', {'penalty': 0, 'C': 1.0}), 0.01), myIntAdd(3, falseBool), myFloatDiv(0.1, myFloatAdd(10.0, 0.1)))	3
AdaBoostLearner(myProd(ARG0, 0), ModifyLearnerFloat(learnerType('LogR', {'penalty': 0, 'C': 1.0}), 0.01), myIntAdd(3, falseBool), myFloatDiv(1.0, 0.1))	3
AdaBoostLearner(myProd(ARG0, 2), ModifyLearnerFloat(learnerType('LogR', {'penalty': 0, 'C': 1.0}), 0.01), myIntAdd(3, falseBool), myFloatDiv(0.1, 0.1))	3
AdaBoostLearner(myProd(ARG0, 1), ModifyLearnerFloat(learnerType('LogR', {'penalty': 0, 'C': 1.0}), 0.01), myIntAdd(3, falseBool), myFloatDiv(0.1, 0.1))	2
AdaBoostLearner(ARG0, learnerType('Boosting', {'learning_rate': 0.1, 'n_estimators': 100, 'max_depth': 3}), 2, 0.1)	2
AdaBoostLearner(myProd(ARG0, 2), ModifyLearnerFloat(learnerType('LogR', {'penalty': 0, 'C': 1.0}), 0.01), myIntAdd(3, falseBool), myFloatDiv(2.1142522088927382, 0.1))	1
AdaBoostLearner(myProd(ARG0, 2), ModifyLearnerFloat(learnerType('LogR', {'penalty': 0, 'C': 1.0}), 0.01), myIntAdd(3, falseBool), myFloatDiv(4.277402487995257, 0.1))	1
AdaBoostLearner(ARG0, learnerType('SVM', {'C': 1.0, 'kernel': 0}), 150, 0.01)	1
AdaBoostLearner(ARG0, learnerType('ExtraTrees', {'n_estimators': 100, 'max_depth': 6, 'criterion': 0}), 2, 0.01)	1

Takeaways

- It's very important to use trial and error when using new programs.
- It is crucial to connect worker nodes to speed up the process of EMADE
- If evaluation function is updated, it is necessary to create a new db as the pareto front of past individuals will not be measured by the same evaluation metrics and may cause errors
- Using grep -rl “error string” path useful to trace the root cause of the error and potentially reduce the amount of individuals with error strings
- Using a mac chip as the master runner gets results pretty quickly