Titanic Machine Learning vs GP

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

9/29/2021

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)
- Extracted the titles from the name and mapped them to integer values
  - {'Mr':1, 'Miss':2, 'Mrs':3, 'Master':4, 'Rare':5}
- 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
- Imputed missing values with the mean of the feature
- Coded all categorical variables (sex, embarked)

Splitting the Data

- We originally split the data into train test split with a test size of 0.2 so that the algorithms can learn on more data and potential perform better but this also made it harder to create the non-dominated pareto front since there is less test data and one more FN or FP can make a big difference
- Used a final split with test size = 0.33
    - Resulted in a higher FNR and FPR but made it easier to create the non-dominated pareto front

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)

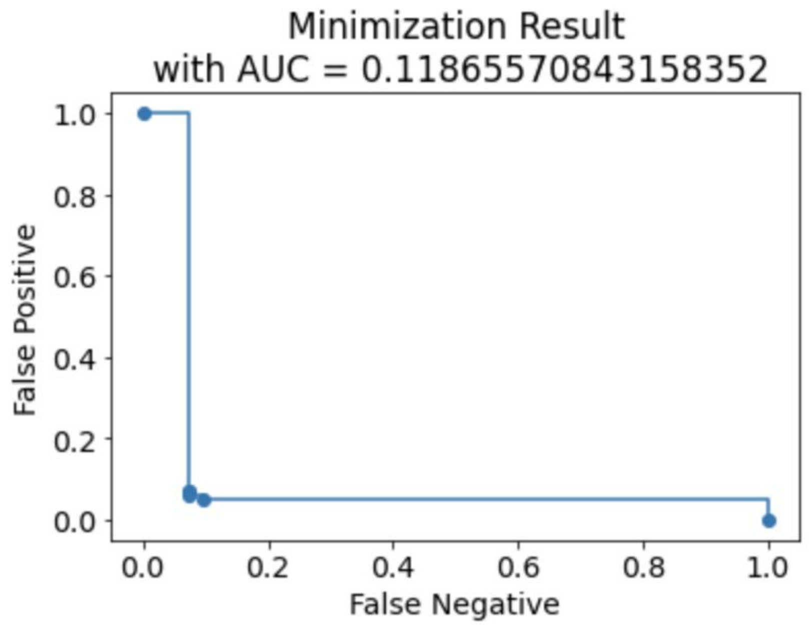
Models

- Multi Layer Perceptron (MLP)
    - decided to use 2 hidden layers with shape (64,32) after testing multiple runs
    - increased learning rate from 0.001 to 0.01 and increased L2 regularization to speed convergence
    - stuck with default optimizer and activation function (“adam”, “relu”)
- Random Forest
    - decided to use the default of 100 trees with a max depth of 3
- Gaussian Naive Bayes
    - used default parameters
- Logistic Regression
    - used default parameters

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

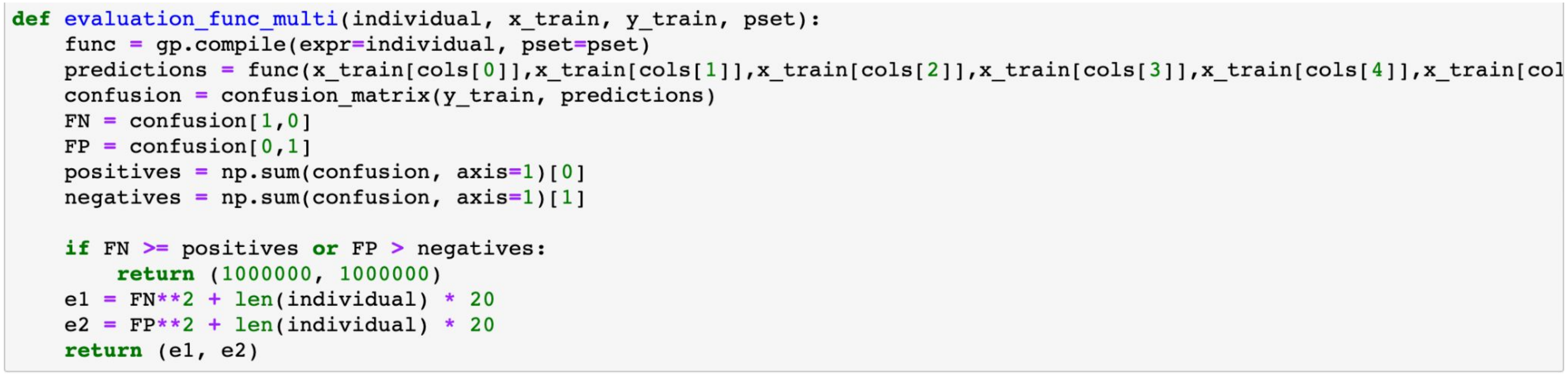
Non-Dominated Pareto Front (Machine Learning)



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
  - To prevent bloating, also added additional tree size * constant to both objectives error
  - 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

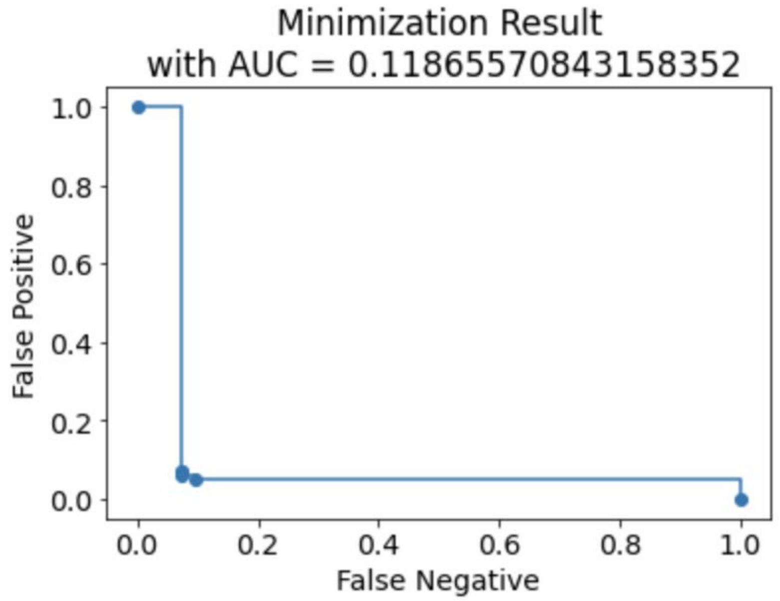
GP Evaluation Function

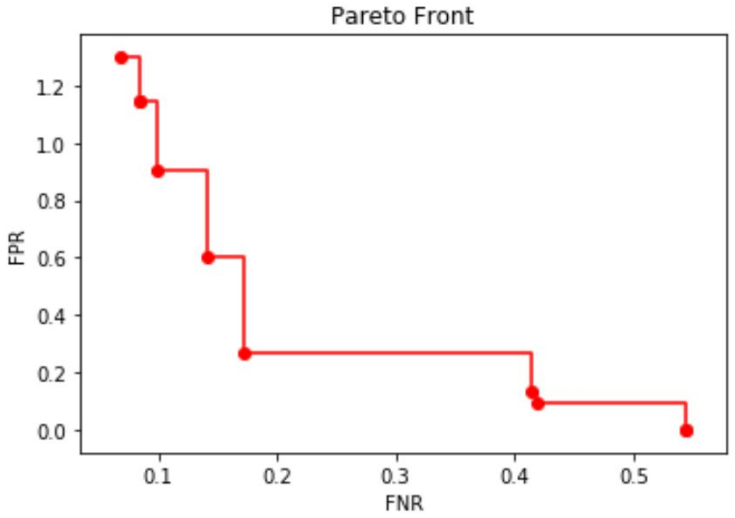


```
def evaluation_func_multi(individual, x_train, y_train, pset):
    func = gp.compile(expr=individual, pset=pset)
    predictions = func(x_train[cols[0]],x_train[cols[1]],x_train[cols[2]],x_train[cols[3]],x_train[cols[4]],x_train[col
    confusion = confusion_matrix(y_train, predictions)
    FN =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

GP Pareto Front vs ML





```
Area Under Curve: 0.1728755376560612
```

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)