Team 1 Midterm Presentation

Aditya Chandaliya, Charlotte Hettrich, Pranav Malireddy, Dhruv Sharma, and Daniel You

10/24/2022

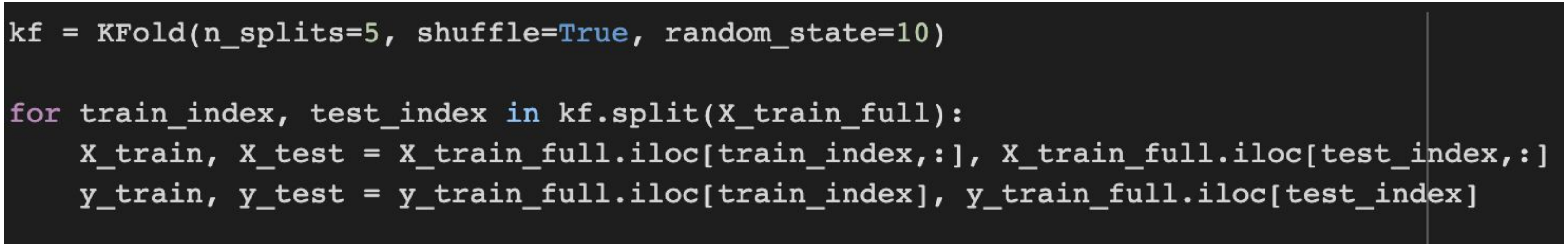
Data Preprocessing

Processing the Data

- Dropped the Name, Ticket, and Cabin columns and set the index to be the PassengerID column on both the train.csv and test.csv dataset
- Replaced the NaN values in several columns:
  - Age column replaced with the mean Age
  - Fare column replaced with the mean Fare
  - Embarked column replaced with the mode value from Embarked
- Changed the strings to integers in the Embarked and Sex columns to make it easier to work with
- Normalized the data
- Folded the data 5 ways using the `KFold` function to split the dataset into 5 consecutive folds

K-Fold Cross-Validator

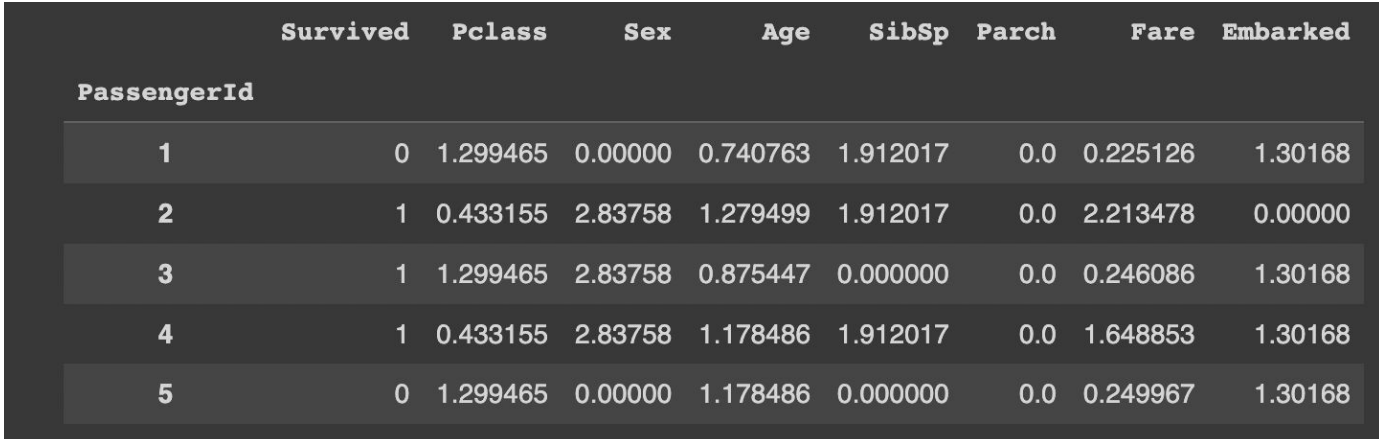
- Provides train and test indices to split data into train and test sets
- Splits the dataset into k consecutive folds
- Each fold is used as a validation while the k-1 remaining folds form the training set
- Parameters:
    - **n_splits**: number of folds
    - **shuffle**: whether to shuffle the data before splitting into batches
    - **random_state**: affects the ordering of the indices, controls randomness of each fold



```
kf = KFold(n_splits=5, shuffle=True, random_state=10)

for train_index, test_index in kf.split(X_train_full):
    X_train, X_test = X_train_full.iloc[train_index,:], X_train_full.iloc[test_index,:]
    y_train, y_test = y_train_full.iloc[train_index], y_train_full.iloc[test_index]
```

Standardization of the Data

- Divide all the numerical values in each column by their mean.
- **Columns**: “Pclass”, “Sex”, “Age”, “SibSp”, “Parch”, “Fare”, and “Embarked”



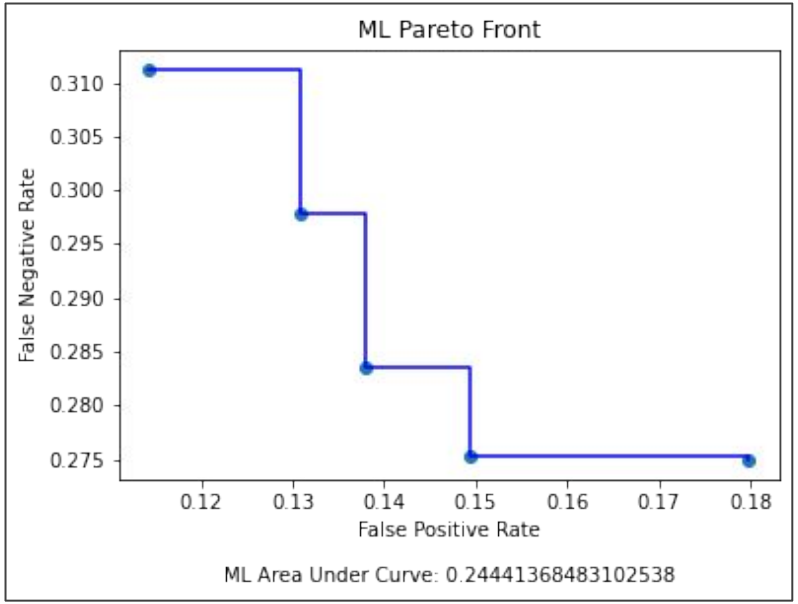
PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
1	0	1.299465	0.00000	0.740763	1.912017	0.0	0.225126	1.30168
2	1	0.433155	2.83758	1.279499	1.912017	0.0	2.213478	0.00000
3	1	1.299465	2.83758	0.875447	0.000000	0.0	0.246086	1.30168
4	1	0.433155	2.83758	1.178486	1.912017	0.0	1.648853	1.30168
5	0	1.299465	0.00000	1.178486	0.000000	0.0	0.249967	1.30168

Titanic ML Algorithms

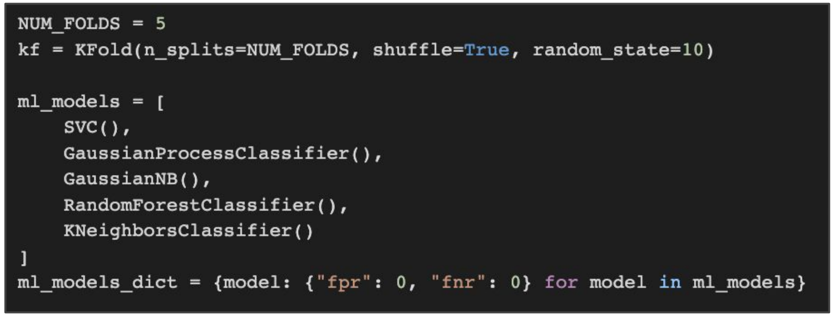
ML Algorithms and Pareto Frontier

- `KNeighborsClassifier()`
- `SVC()`
- `GaussianNB()`
- `GaussianProcessClassifier()`
- `RandomForestClassifier()`

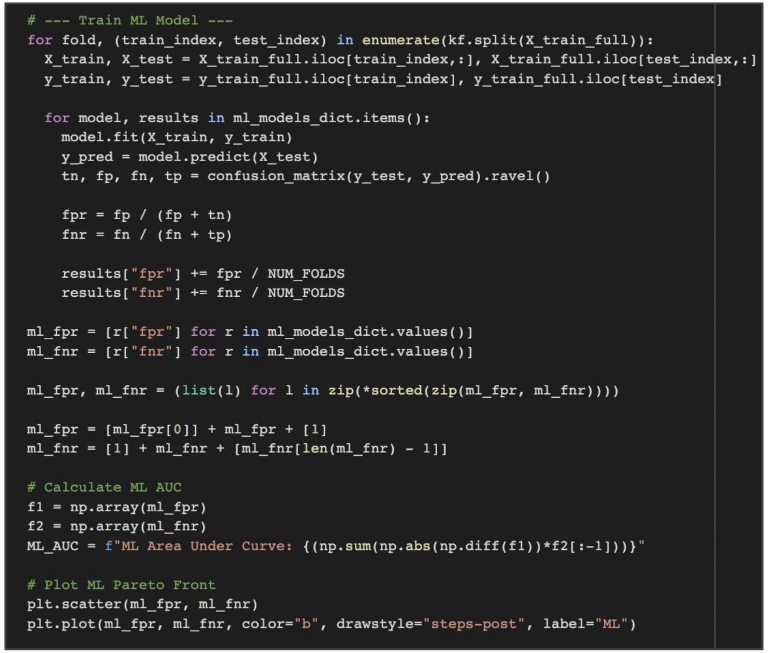
- The Pareto front to the right denotes average FNR and FPR over the 5 folds.
- We had to change the ML Algorithms due to the normalization of the data.



Graphing ML Pareto Front



```python
NUM_FOLDS = 5
kf = KFold(n_splits=NUM_FOLDS, shuffle=True, random_state=10)

ml_models = [
    SVC(),
    GaussianProcessClassifier(),
    GaussianNB(),
    RandomForestClassifier(),
    KNeighborsClassifier()
]
ml_models_dict = {model: {"fpr": 0, "fnr": 0} for model in ml_models}
```



```python
# --- Train ML Model ---
for fold, (train_index, test_index) in enumerate(kf.split(X_train_full)):
  X_train, X_test = X_train_full.iloc[train_index,:], X_train_full.iloc[test_index,:]
  y_train, y_test = y_train_full.iloc[train_index], y_train_full.iloc[test_index]

  for model, results in ml_models_dict.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

    fpr = fp / (fp + tn)
    fnr = fn / (fn + tp)

    results["fpr"] += fpr / NUM_FOLDS
    results["fnr"] += fnr / NUM_FOLDS

ml_fpr = [r["fpr"] for r in ml_models_dict.values()]
ml_fnr = [r["fnr"] for r in ml_models_dict.values()]

ml_fpr, ml_fnr = (list(l) for l in zip(*sorted(zip(ml_fpr, ml_fnr))))

ml_fpr = [ml_fpr[0]] + ml_fpr + [1]
ml_fnr = [1] + ml_fnr + [ml_fnr[len(ml_fnr) - 1]]

# Calculate ML AUC
f1 = np.array(ml_fpr)
f2 = np.array(ml_fnr)
ML_AUC = f"ML Area Under Curve: {(np.sum(np.abs(np.diff(f1))*f2[:-1]))}"

# Plot ML Pareto Front
plt.scatter(ml_fpr, ml_fnr)
plt.plot(ml_fpr, ml_fnr, color="b", drawstyle="steps-post", label="ML")
```

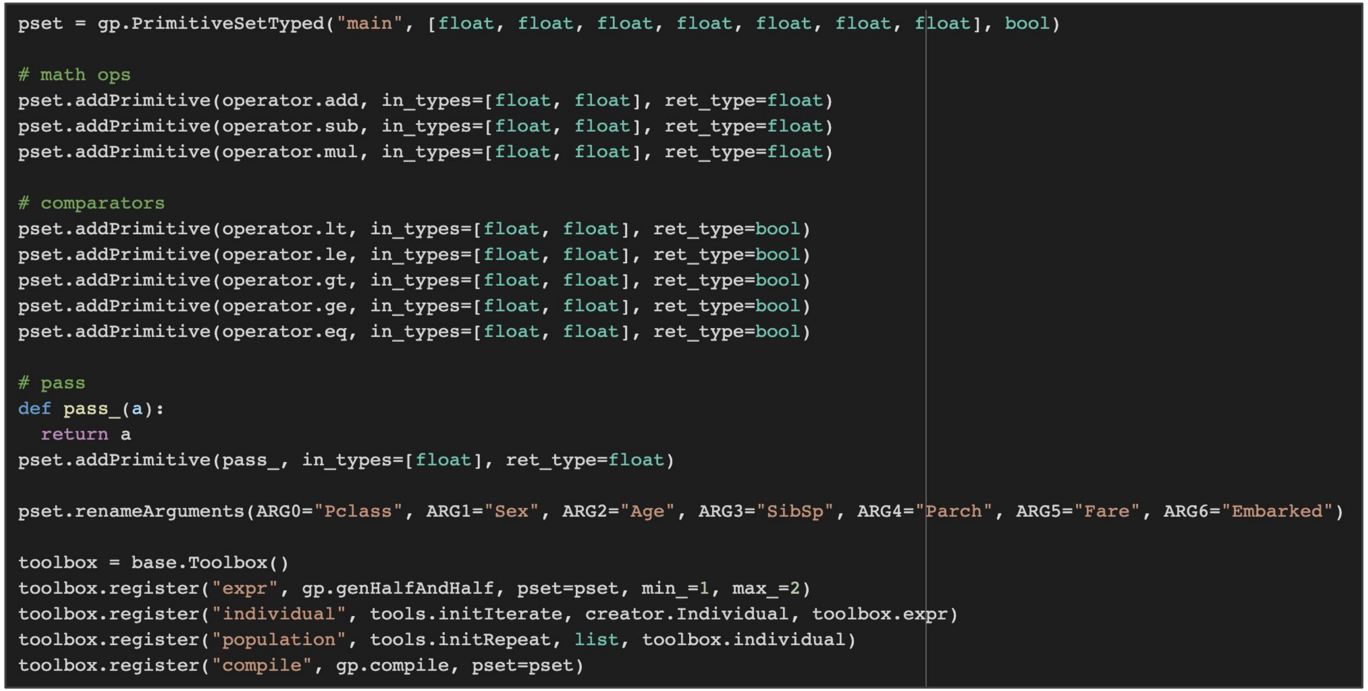
Titanic MOGP

Initializing our Primitives & Toolbox

- Decided to use strongly-typed GP (7 Floats -> Bool)
- Used add, sub, mul, neg, lt, gt, eq primitives (from builtin operator package)
- Created a `pass_(a)` function to add into primitive set

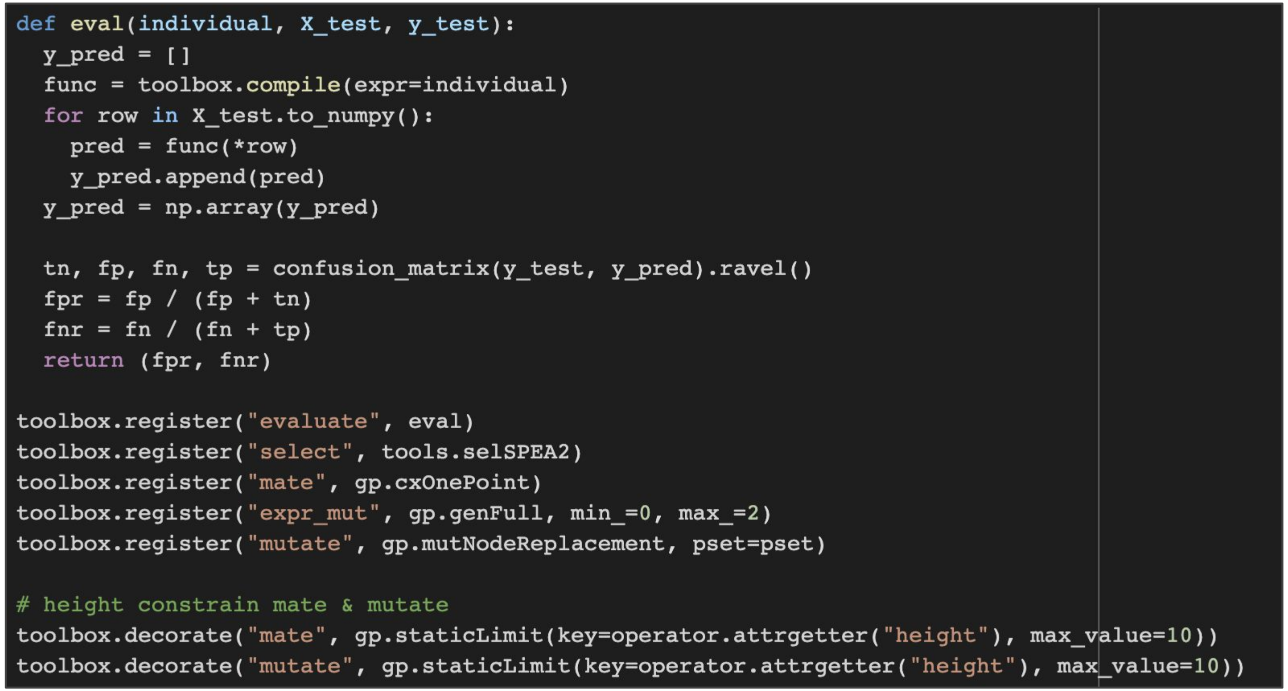
- For mating, we used `deap.gp.cxOnePoint`
- For mutation, we used `deap.gp.mutNodeReplacement`
- We height-constrained both mating and mutation to 10
- For selection, we used `deap.tools.selSPEA2`

Primitive Set Creation



```python
pset = gp.PrimitiveSetTyped("main", [float, float, float, float, float, float, float], bool)

# math ops
pset.addPrimitive(operator.add, in_types=[float, float], ret_type=float)
pset.addPrimitive(operator.sub, in_types=[float, float], ret_type=float)
pset.addPrimitive(operator.mul, in_types=[float, float], ret_type=float)

# comparators
pset.addPrimitive(operator.lt, in_types=[float, float], ret_type=bool)
pset.addPrimitive(operator.le, in_types=[float, float], ret_type=bool)
pset.addPrimitive(operator.gt, in_types=[float, float], ret_type=bool)
pset.addPrimitive(operator.ge, in_types=[float, float], ret_type=bool)
pset.addPrimitive(operator.eq, in_types=[float, float], ret_type=bool)

# pass
def pass_(a):
  return a
pset.addPrimitive(pass_, in_types=[float], ret_type=float)

pset.renameArguments(ARG0="Pclass", ARG1="Sex", ARG2="Age", ARG3="SibSp", ARG4="Parch", ARG5="Fare", ARG6="Embarked")

toolbox = base.Toolbox()
toolbox.register("expr", gp.genHalfAndHalf, pset=pset, min_=1, max_=2)
toolbox.register("individual", tools.initIterate, creator.Individual, toolbox.expr)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("compile", gp.compile, pset=pset)
```

Evaluation

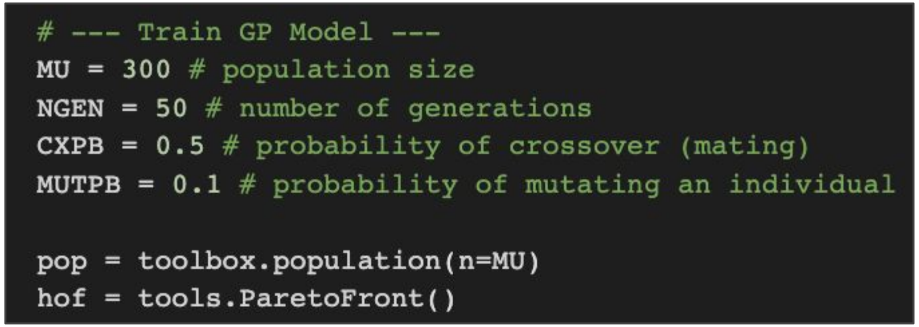


```python
def eval(individual, X_test, y_test):
  y_pred = []
  func = toolbox.compile(expr=individual)
  for row in X_test.to_numpy():
    pred = func(*row)
    y_pred.append(pred)
  y_pred = np.array(y_pred)

  tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
  fpr = fp / (fp + tn)
  fnr = fn / (fn + tp)
  return (fpr, fnr)

toolbox.register("evaluate", eval)
toolbox.register("select", tools.selSPEA2)
toolbox.register("mate", gp.cxOnePoint)
toolbox.register("expr_mut", gp.genFull, min_=0, max_=2)
toolbox.register("mutate", gp.mutNodeReplacement, pset=pset)

# height constrain mate & mutate
toolbox.decorate("mate", gp.staticLimit(key=operator.attrgetter("height"), max_value=10))
toolbox.decorate("mutate", gp.staticLimit(key=operator.attrgetter("height"), max_value=10))
```

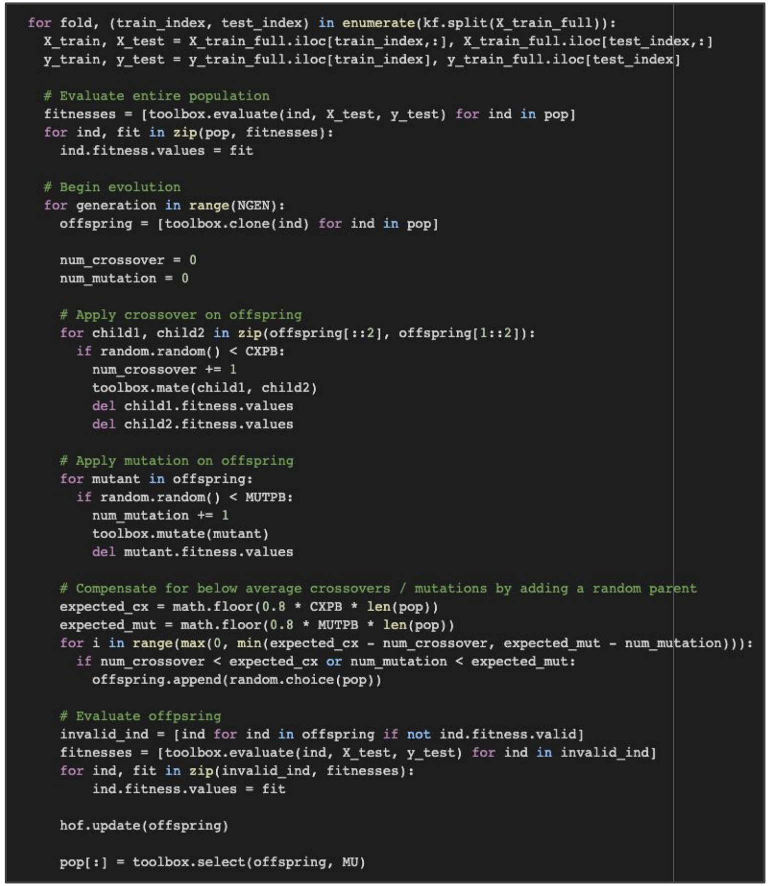
Design of Evolutionary Algorithm

- To create the algorithm, we consulted code from Lab 1 and 2 as well as the algorithms.py file under the DEAP documentation
- We noticed that the `varOr()`utility involved the idea of reproduction, so we decided to incorporate this in our model
- The idea behind this reproduction component is that if there were a lack of crossover/mutation in a certain generation, we could compensate by adding a random parent to the offspring

Evolutionary Algorithm

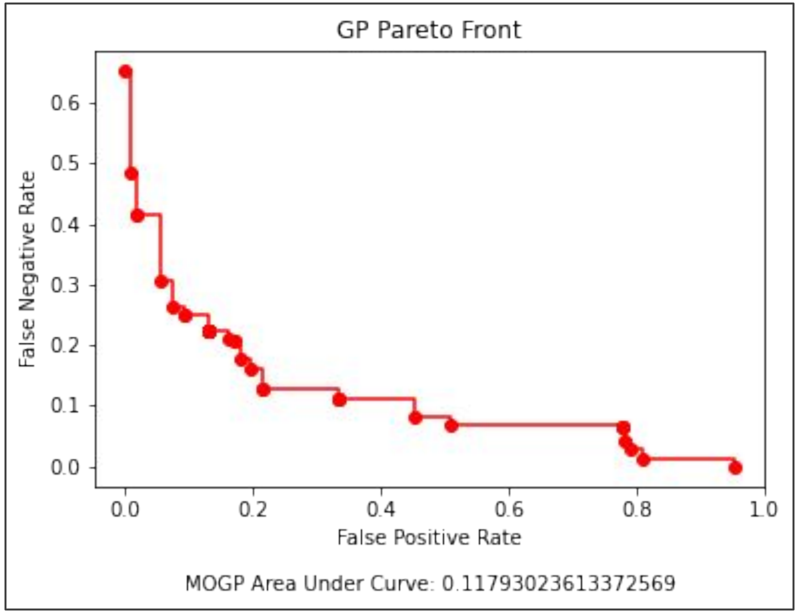


```python
# --- Train GP Model ---
MU = 300 # population size
NGEN = 50 # number of generations
CXPB = 0.5 # probability of crossover (mating)
MUTPB = 0.1 # probability of mutating an individual

pop = toolbox.population(n=MU)
hof = tools.ParetoFront()
```



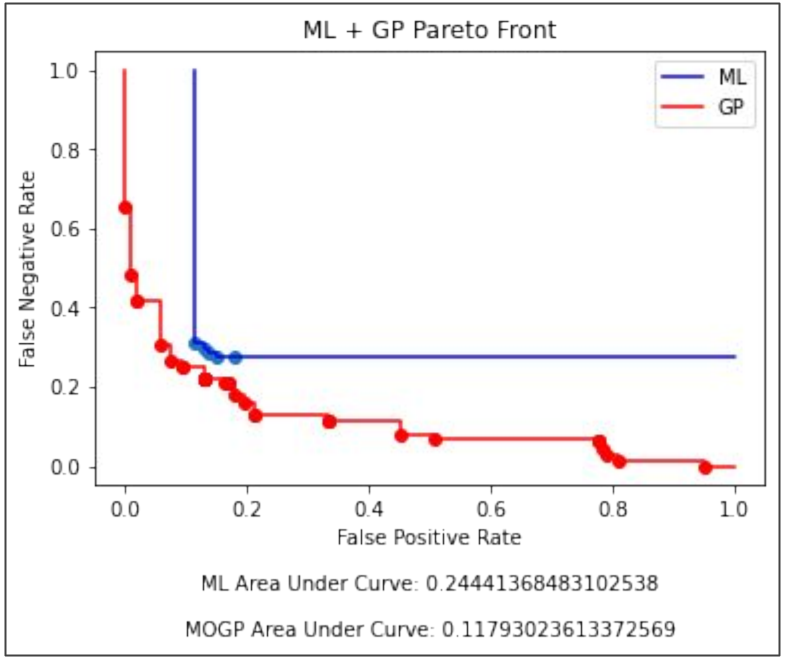
```python
for fold, (train_index, test_index) in enumerate(kf.split(X_train_full)):
  X_train, X_test = X_train_full.iloc[train_index,:], X_train_full.iloc[test_index,:]
  y_train, y_test = y_train_full.iloc[train_index], y_train_full.iloc[test_index]

  # Evaluate entire population
  fitnesses = [toolbox.evaluate(ind, X_test, y_test) for ind in pop]
  for ind, fit in zip(pop, fitnesses):
    ind.fitness.values = fit

  # Begin evolution
  for generation in range(NGEN):
    offspring = [toolbox.clone(ind) for ind in pop]

    num_crossover = 0
    num_mutation = 0

    # Apply crossover on offspring
    for child1, child2 in zip(offspring[::2], offspring[1::2]):
      if random.random() < CXPB:
        num_crossover += 1
        toolbox.mate(child1, child2)
        del child1.fitness.values
        del child2.fitness.values

    # Apply mutation on offspring
    for mutant in offspring:
      if random.random() < MUTPB:
        num_mutation += 1
        toolbox.mutate(mutant)
        del mutant.fitness.values

    # Compensate for below average crossovers / mutations by adding a random parent
    expected_cx = math.floor(0.8 * CXPB * len(pop))
    expected_mut = math.floor(0.8 * MUTPB * len(pop))
    for i in range(max(0, min(expected_cx - num_crossover, expected_mut - num_mutation))):
      if num_crossover < expected_cx or num_mutation < expected_mut:
        offspring.append(random.choice(pop))

    # Evaluate offpsring
    invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
    fitnesses = [toolbox.evaluate(ind, X_test, y_test) for ind in invalid_ind]
    for ind, fit in zip(invalid_ind, fitnesses):
        ind.fitness.values = fit

    hof.update(offspring)

    pop[:] = toolbox.select(offspring, MU)
```

MOGP Pareto Frontier



MOGP Area Under Curve: 0.11793023613372569

Comparison of ML & MOGP Results



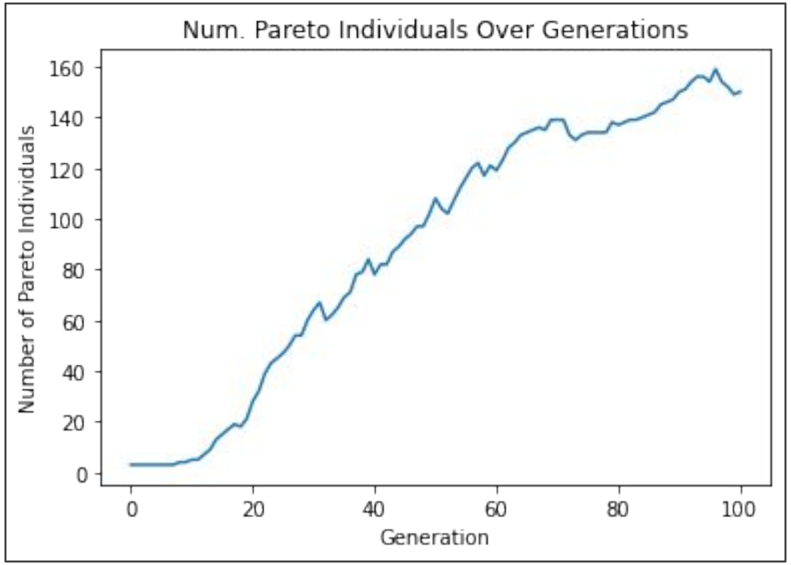
ML Area Under Curve: 0.24441368483102538

MOGP Area Under Curve: 0.11793023613372569

Titanic EMADE

Using EMADE

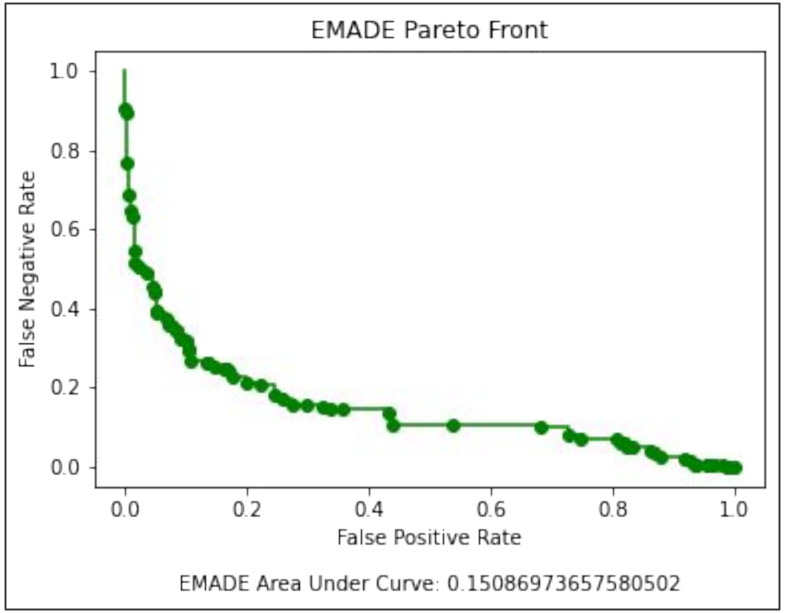
- We were able to connect to Daniel's master process via his IP address.
- Most of us used SQL Workbench to query the data, and see the rows in the tables.
- We ran for 100 generations overnight as a cluster of computers.
- We found that the number of Pareto individuals increased as the number of generations increased, generation 0 only had 3 Pareto individuals while generation 100 had 150 Pareto individuals.



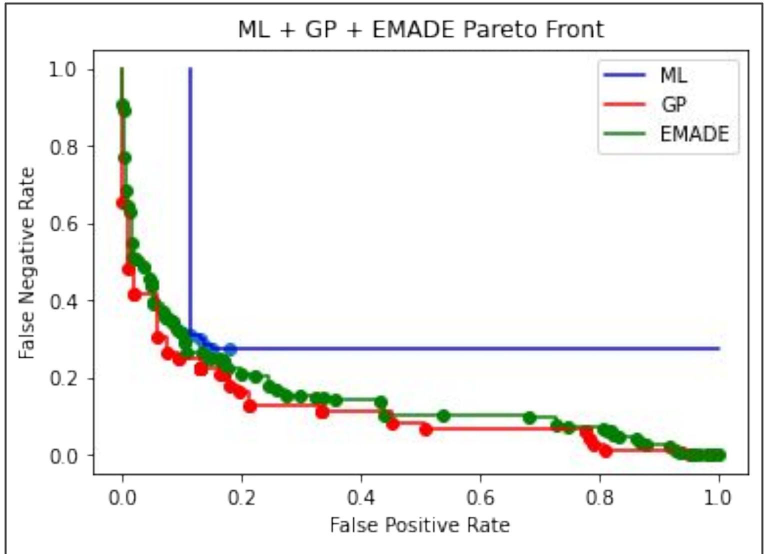
Using EMADE (cont.)

- Created a script to apply our preprocessing steps on `train.csv` to create a set of `.csv.gz` files for EMADE
- Used the default settings found in `input_titanic.xml`
- Obtained Pareto front using SQL queries to match the hash in the `paretofront` table with the individual in the `individuals` table
- EMADE outputted False Positive and False Negative counts:
    - To make this comparable with our data, we averaged the total positive and total negative counts over 5 folds, which were:
        - Total Positive = 68.4
        - Total Negative = 109.8
    - Then, we divided what EMADE outputted with the above numbers

EMADE Pareto Frontier



Comparison of ML, MOGP, and EMADE Results



ML Area Under Curve: 0.24441368483102538

MOGP Area Under Curve: 0.11793023613372569

EMADE Area Under Curve: 0.15086973657580502

Conclusion

EMADE Problems

- Setting up EMADE was different on every computer
  - MySQL
  - Python Version + Libraries
- Different Python versions led to issues with the `pickle` Python package, as pickle protocol 5 is used for Python 3.8+
- Must be very careful running EMADE with reuse = 0, as it will overwrite the database

Takeaways

- Throughout the process we learned how to use scikit-learn and how to use different Machine Learning algorithms with a large data set.
- It takes multiple tries to find Machine Learning algorithms that are codominant with the false positive and false negative rates especially with normalization of the data.
- EMADE outputted a lot more data than what we could analyze in time, which is something we can definitely improve for the future.