Nature Inspired Computing - CA2 - Individual Report

1. Contribution

After going through a lot of options, considering various options like knowledge and skill of all the teammates and amount of time we have, we chose the GECCO 2019 project "Internet of Things: Online Event Detection for Drinking Water Quality Control". After going through the problem statement and dataset, we realised that we have enough time to try various approaches to solve the problem. To solve the problem, I chose the following nature-inspired approaches: Differential Evolution and Genetic Algorithm. To handle the class imbalance, I researched a few methods and experimented with various techniques like SMOTE and RUS, some of which are briefly included in this report.

2. Algorithm Survey

While conducting a preliminary exploratory data analysis, we've noticed a huge class imbalance in our dataset. Our aim is to build a robust classifier that can handle the class imbalance. So, our research also included techniques to handle class imbalance along with various algorithms used in building the classifier. I proposed the following algorithms – Genetic Algorithm, Differential Algorithm using various classification algorithms like Random Forest, Logistic Regression and Multilayer Perceptron. As a part of handling class imbalance, our research also included techniques to handle class imbalance.

Handling Class Imbalance:

Out of 132,480 records in our dataset, more than 99.84%(132,268) of the target variable is a non-event and only 0.0016% (212) is the event class(anomaly) that needs to be predicted/detected. Various techniques like SMOTE, RUS have been tried and SMOTE is shown to have given the best performance when tested on heuristic classifiers. Below is its implementation:

```
1. from imblearn.over_sampling import SMOTE
2. from sklearn.model_selection import train_test_split
3. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
4. sm = SMOTE(random_state=2)
5. X_train_res, y_train_res = sm.fit_resample(X_train, y_train.ravel())
```

Note: Random oversampled data using SMOTE has been created based on the train data only. **As a best practice, the actual test data is set aside as it is and used only for validation purposes.**

Building Classifier: Our approach is to use nature-inspired algorithms to perform hyperparameter tuning of various classification models and build a robust anomaly detector. In general, Grid Search and Random Search are two of the most common hyperparameter optimisation algorithms in classical Machine Learning. Instead, we used Differential Evolution and Genetic Algorithms as optimization procedures to perform the hyperparameter optimisation (tuning). **In the n-dimensional search space, each dimension represents a hyperparameter and each point represents a model configuration.** The scale of the dimension is the value that the hyperparameter takes– real, integer or categorical.

A. **Genetic Evolutionary Algorithm:** We used the Genetic Algorithm as an approach to optimise the hyperparameters in Random Forest Classifier(RFC). We've selected six crucial hyperparameters some of which are categorical, and others are continuous. A reasonable numerical range has been fixed to all the continuous-valued hyperparameters and all categorical values have been label-encoded in such a way that each of its possible values is represented by an integer and the algorithm would randomly choose one of those values. For example, a hyperparameter called

bootstrap could take only True or False as its values. So, those values would be encoded and represented as 1 and two respectively.

Representation: Each **chromosome is represented as a vector of length six** (having 6 dimensions in solution space) each of which represents the value of the six hyperparameters. Below is the list of hyperparameters, their possible values and their python representations

1	2	3	4	5	6
bootstrap	max_depth	max_features	min_samples_leaf	min_samples_split	n_estimators
True, False	[1,100]	auto, sqrt, log2	[1,5]	[2,10]	[1, 2000]

Note: Any [x, y] *indicates that the corresponding hyperparameter can hold any value between x and y.*

```
1. bootstrap_lookup = {1: True, 2: False}
2. max_depth_lookup = np.arange(1, 100)
3. max_features_lookup = {1: 'auto', 2: 'sqrt', 3: 'log2'}
4. min_samples_leaf_lookup = np.arange(1,5)
5. min_samples_split_lookup = np.arange(2,10)
6. n_estimators_lookup = np.arange(1,2000)
```

For this competition, the quality of the anomaly detector is calculated using the F1 score, the harmonic mean of precision and recall. So, the **F1 score is used as the fitness function**. A population of 10 chromosomes has been created in which each chromosome is a six-dimensional vector with values randomly chosen from the above possibilities. Then the below Genetic Algorithm is implemented.

Algorithm:

```
1. Generate a population of p randomly generated solutions and calculate fitness values.
2. Choose any two parents from the above population using Binary Tournament
3. Using Single point crossover on the above-selected parents and create two new children
4. Using Mutation, randomly change the value of any gene in both the children and create
   two new children.
5. Calculate the fitness of the above-generated children and run weakest replacement on
   the population using the mutated children.
6. Run the above steps until the termination criterion has been reached.
```

The above algorithm is run for 10 generations after which the final maximum F1 score **is** 0.8907 while the initial score is 0.8148.

Drawbacks: The **time-complexity of the Genetic Algorithm approach is very high** – the fitness of a child can only be measured after a new classifier is built and the F1 score is calculated. So, once every generation, two new Random Forest classifiers are built which is quite expensive computationally.

B. Differential Evolutionary Algorithm:

Differential Evolution is a heuristic approach for the global optimisation of continuous space functions. It optimizes a function by iteratively trying to improve a candidate solution with regards to a given measure of quality. A DE is often considered a subset of the broader space of genetic algorithms but with a few restrictions: a. The genotype is a real-valued vector b. The crossover and mutation operators use the difference between two or more vectors in the population to create a new vector. In Mutation, individuals are generated as follows:

$$X_{r1,G+1} = X_{r1,G+1} + F * (X_{r2,G} - X_{r3,G})$$

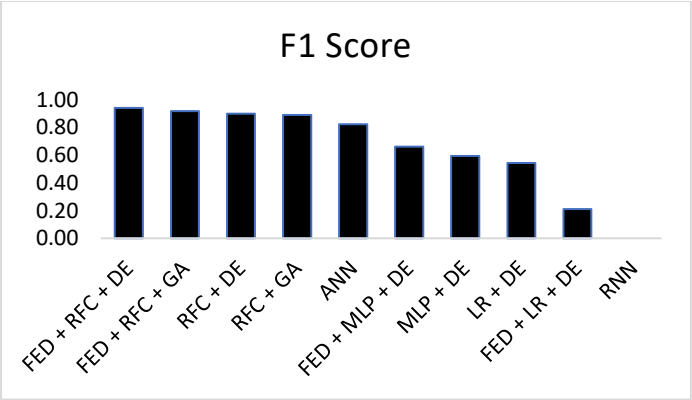
Where r1, r2, r3 are random numbers generated within a specific interval[1, NP] and the Variation factor F is a real number of the interval [0, 2]. Here, NP is the size of the population.

3. Experimentation

Using SciPy's Differential Evolution(DE) implementation, Logistic Regression, Random Forest Classifier and Multi-Layer Perceptron are optimised based on the F1 score of those models as the fitness value of the function to be optimised. **SciPy's DE implementation is configured to minimise a function**, but our objective is to maximise the F1 score. **Hence, we're minimising 1-f1 which is the same as maximising F1.** Below is a snippet of my code where an MLP Classifier is being optimised using Differential Evolution.

```
1.  def get_mlp_model(x): # x is a vector that contains random values for hyperparameters
2.      model = MLPClassifier(activation='tanh', solver='adam', alpha=x[0],
3.                            hidden_layer_sizes=(int(x[1]), int(x[2])),random_state=1)
4.      model.fit(X_train, y_train)
5.      return 1 - f1_score(y_test, y_pred)
6.
7.  MLP_bounds = [(0, 1e-1), (1, 100), (1, 10)] # Bounds for MLP Classifier
8.  MLP_result = differential_evolution(func = get_mlp_model, bounds = MLP_bounds, maxiter
    = 10, popsize = 10)
```

Similarly, other classification models like Logistic Regression and Random Forest are optimised using DE. One of my teammates worked on Feature Engineering and created new features like **rolling median, rolling minimum, and rolling maximums** for all the existing features. So, all the algorithms were run on both data sets – the actual dataset and the newly created **Feature Engineering Dataset**.



- **ANN - Artificial Neural Network**
- DE - Differential Evolution
- FED - Feature Engineered Dataset
- GA - Genetic Algorithm
- LR - Logistic Regression
- MLP - Multi-Layer Perceptron
- RFC - Random Forest Classifier
- **RNN - Recurrent Neural Network**

The above bar chart depicts the values of F1-scores of models built using several optimisation techniques. Results highlighted in orange are F1 scores of the algorithms implemented by my teammates. Below is a table depicting various metrics of all the models built.

Algorithm	Accuracy	Precision	Recall	F1 Score
FED + RFC + DE	0.9998	0.9286	0.9559	0.9420
FED + RFC + GA	0.9997	0.9256	0.9134	0.9195
RFC + DE	0.9997	0.8571	0.9474	0.9000
RFC + GA	0.9997	0.9464	0.8413	0.8908
ANN	**0.9990**	**0.9520**	**0.7280**	**0.8250**
FED + MLP + DE	0.9989	0.6508	0.6721	0.6613
MLP + DE	0.9978	0.8714	0.4519	0.5951
LR + DE	0.9986	0.5397	0.5484	0.5440
FED + LR + DE	0.9875	0.8571	0.1198	0.2102
RNN	**0.9988**	**0.0000**	**0.0000**	**0.0000**

- **ANN - Artificial Neural Network**
- DE - Differential Evolution
- FED - Feature Engineered Dataset
- GA - Genetic Algorithm
- LR - Logistic Regression
- MLP - Multi-Layer Perceptron
- RFC - Random Forest Classifier
- **RNN - Recurrent Neural Network**

Our Proposed Best Classifier

Although several models performed better than our proposed best model - ANN, we did not put them forth as our final solutions because they're classical Machine Learning classifiers at their core i.e.,

they're not fully nature-inspired. Hence, we chose ANN as the best model as it's not only performing decently but also fully nature inspired. Below is the list of various versions of algorithms implemented and their accuracy metrics. With an F1 score of 0.9420, the Random Forest classifier built using Feature Engineered Data and optimised using Differential algorithm has performed the best.

4. Teamwork and Conclusions:

After we finalised the problem, it took a while for us to fully commit ourselves to this coursework as we were stuck with a few other deadlines. Nevertheless, we still had regular meetings to communicate and share whatever little progress we made. But once we started focusing on this project, we were able to finish the project quickly than expected. I should stress on the fact that we not only took our skillset into consideration but also out interests while choosing our approaches towards solving the problem. In the team meetings, we proposed various approaches of our interest and got constructive feedback from the team.

Our team consists of four members – Aayush, Andrew, Manoj and Yasaswi(me). While Aayush chose Artificial Neural Networks and worked on implementing that, Andrew chose Recurrent Neural Network and worked towards implementing it. I chose Differential Evolution and Genetic Algorithm and tried to build the anomaly detector using these algorithms. Manoj worked on feature engineering, created the readme and MOM files. I should also note that 3 out of 4 of our teammates(except Manoj) are considerably very good at coding. Hence, we were mostly working on algorithmic approaches while he was doing the research and writing part.

As a team, we realised most of the pros and cons of our approaches even before we started coding. We created a group in WhatsApp to update each other about our progress and ask for help if need be and that proved a lot useful. As each of us were working towards the solution using algorithms that excite us personally, we were able to finish the project quicker than expected leaving us with some spare time in hands. Each of us tried one of the approaches aiming to choose the best one finally and all the team collaborated equally towards the project with their specific expertise.

I should mention that the project gave me an opportunity to collaborate with students from diverse backgrounds and learn a lot from them – including but not limited to presenting my findings in a neat and orderly fashion, using LaTeX to write a report, time management and many more.

5. Conclusion

Using an ANN, we've successfully solved the GECCO 2019 problem of developing an anomaly detector. The networks' outer layer bias was manually set to overcome the hardships that arise when dealing with a high 'class imbalance'. With hyper-parameter tuning, we have managed to achieve this with a performance score of 82.5% (using the F1-Score metric). Amongst the approaches like ANN, RNN and DE that we currently tried, the performance can be improved by handling class imbalance more sensibly using various techniques instead of just SMOTE – one such technique is a unique oversampling method using genetic crossover and mutation operators where the newly created records are labelled using a distance-based classifier like KNN. There're other novel approaches like multi-objective optimisation, PSO and artificial immune systems that can be used to solve this problem. With more time or more team members, we could've been able to create better detection systems using other nature-inspired algorithms.

6. References:

https://bit.ly/3lR8ome https://bit.ly/30arOea https://bit.ly/3DMnakw
https://bit.ly/3GlXQU2 https://bit.ly/3pM9d0Z https://bit.ly/31Sn79y