BBM467 - Small Data Science Project

Budgerigar

1st Student: 21946606, Mert Tazeoğlu
2nd Student: 21827517, Yusuf Efe Kalaycı

Optimization Techniques In Machine Learning

Optimization is a process in order to find a method or a set of inputs to an objective function that results in a minimum or maximum function evaluation. Optimization is not only important but also challenging problem especially in machine learning. Because from training artificial netural Networks to fitting regression models, different studies requires different features. Thats why of optimization is at the heart of machine learning.

In 21st century, coding is not just coding since other things such as optimization also matters. Most of programmers can write code easily, right? But how many of programmers can write optimized code? Thats where of optimization shows up. We can say that almost every engineering product is a form of a solution of an optimization problem. For example, in machine learning optimization is used for several reasons:

- Minimizing error, cost or loss when fitting algorithm with simplifying dataset and app
- Reducing computational load for procesisng large datasets with using data effectively
- Providing better performance and accuracy rates for complex multi-dimensional spaces

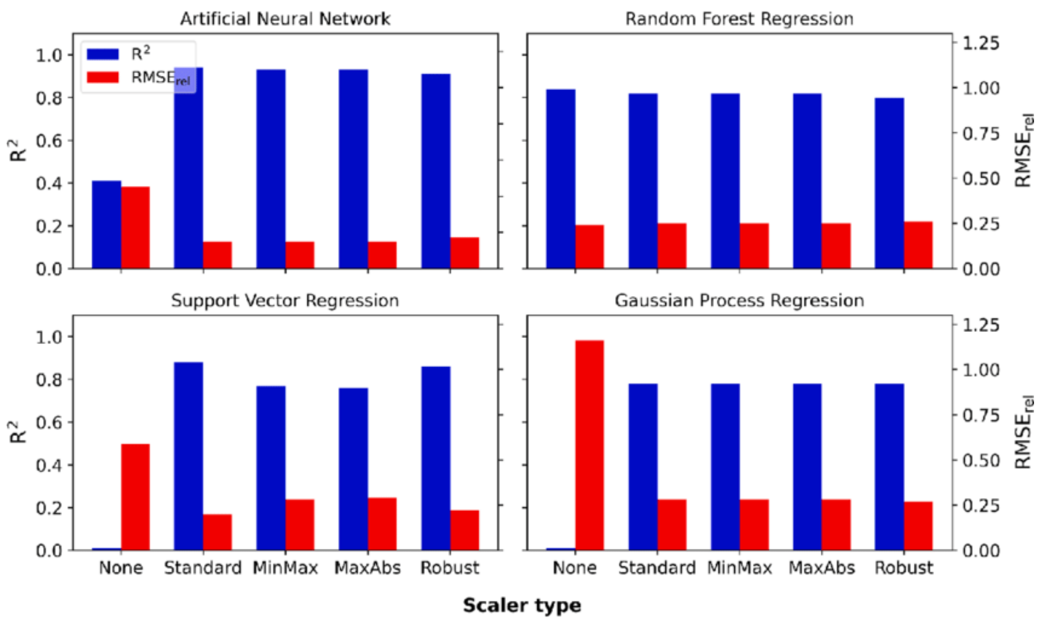


Figure-1: Importance of Optimization In ML (Example) [1]

1- Scaling Features

Feature scaling is a technique to standardize the independent features present in the data in a fixed range. [2] Scaling is performed during data preprocessing stage in order to take care of highly varying magnitudes or values or units. It this kind features don’t be scaled, then machine learning algorithm will tend to perceive greater values as higher and small values as lower since it doesn’t care about unit of values. For example when comparing 1.6 m and 87 cm, algorithm will make mistake. With feature scaling, values will be in similar range therefore they will be easier to compare. Also they will have similar influence on analytical model.

① $$x_{new} = \frac{x_{old}}{x_{max}}$$

Simple Feature Scaling

② $$x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}$$

Min-Max Scaling

③ $$x_{new} = \frac{x_{old} - \mu}{\sigma}$$

Z-Score Scaling

Figure-2: Different Scaling Methods

```
# Scaling Data In Python
import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df = pandas.read_csv(“simple_df.csv”)
X = df[[“Column_1”,“Column_2”]]

scaled_X = scale.fit_transform(X)
print(scaled_X)

# Scaling is also can be done without using libraries
```

```
# Output:
[[-2.05595938 -1.36346644]
 [-0.55826591 -0.35428562]
…
 [0.43029311 2.16725135]]
```

2- Exhaustive Search

Exhaustive search (in other words brute force) is process of looking most optimal hyperparameters by checking whether each candidate is good match. [3] Brute force guarantees to find solutions by listing and checking all of possibilities. Also it is a generic method, easy to implement and understand. For these reasons exhaustive search is ideal to use when solving smaller and simpler problems. But it should’t be forget that, brute force algorithms are getting really slower (with O(n!) growth) when input size increases. Brute force is not efficient because it is possible to find better solutions with smarter designs.

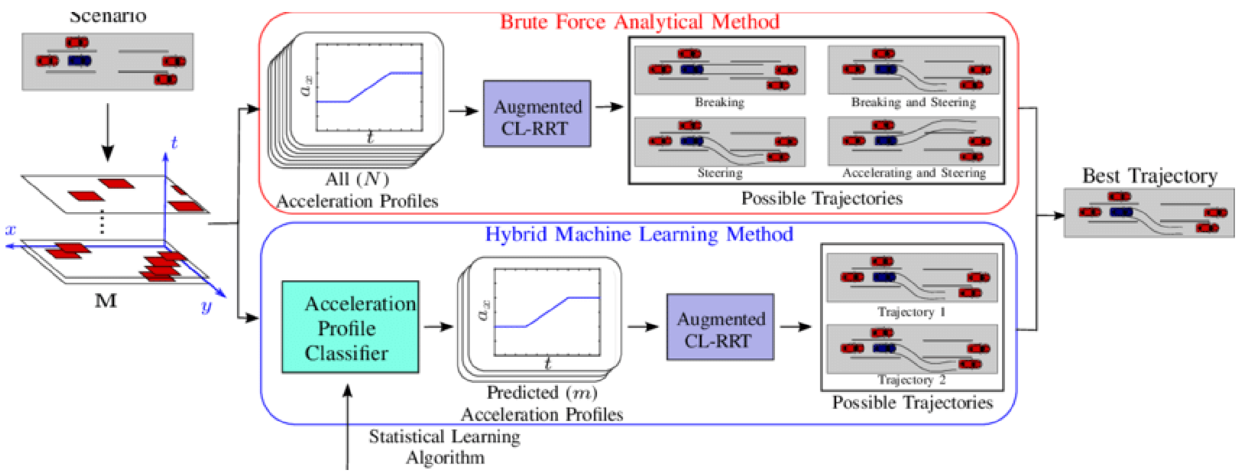


Figure-3: An Example Of Brute Force Application In Machine Learning [4]

Exhaustive search can be used for several goals in machine learning. For example, in a non-linear regression project finding T in y=O^TX can be done with brute force approach. Or in KNN algorithm, if K will be low then model will be overfitted and if K will be high then model will be underfitted. There finding best value is important and best value of K can be find with using brute force approach.

```
# Brute Force Approach In Python (Method-1)

import numpy as np
from sklearn.neighbors import KNeighborsClassifier

neighbors = np.arange(1,8)
train_accuracies = {}
test_accuracies = {}

for neighbor in neighbors:
    knn = KNeighborsClassifier(n_neighbors=neighbor)
    knn.fit(X_train, y_train)
    train_accuracies[neighbor] = knn.score(X_train, y_train)
    test_accuracies[neighbor] = knn.score(X_test, y_test)

print(neighbors, '\n', train_accuracies, '\n', test_accuracies)

# Larger K = Less Complex Model = Can Cause Underfitting
# Smaller K = More Complex Model = Can Lead To Overfitting
# With using these codes, we will be able to see ideal K value.
```

```
# Output:
[1,2,3,4,5,6,7]
{1:1.00, 2:0.88, 3:0.90, 4:0.87, 5:0.88, 6:0.86, 7:0.87}
{1:0.78, 2:0.85, 3:0.84, 4:0.85, 5:0.85, 6:0.86, 7:0.86}
```

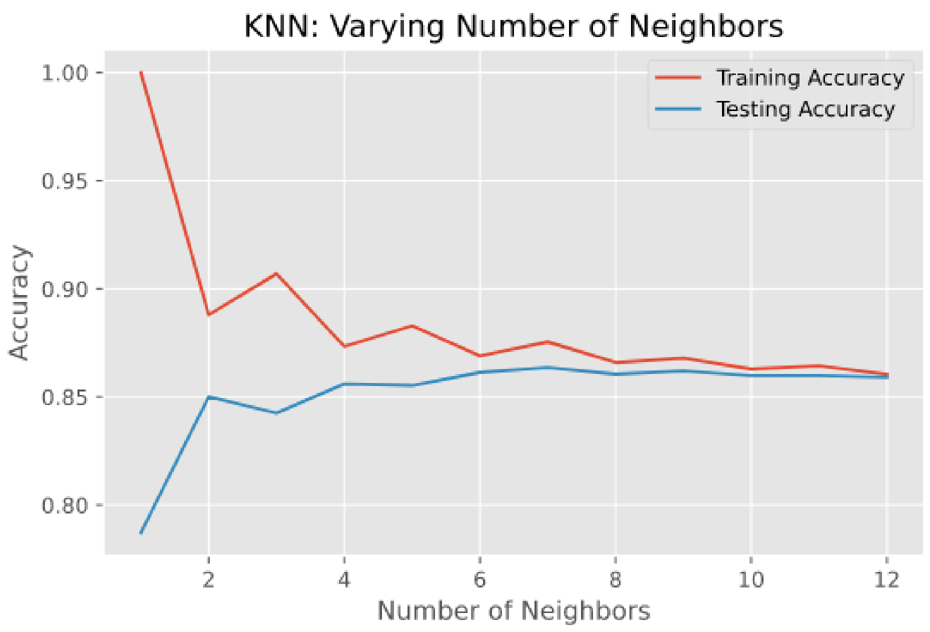


Figure-4: KNN Model Performance Evaluation With Brute Force Approach

```
# Brute Force Approach In Python (Method-2)[5]

# This also can be done with using a better way, which is Sklearn’s
GridSearchCV module.

import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import GridSearchCV

# Only odd numbers to prevent ties.
param_grid = {‘n_neighbors’: np.arange(1,12,2)}
knn = KNeighborsClassifier()

ss = StratifiedShuffleSplit(n_splits=5, test_size=.25, random_state=0)
gscv = GridSearchCV(knn, param_grid, cv=ss)
gscv.fit(X,y)

print(‘Best Params:, gscv.best_params_)
print(‘Best Score:, gscv.best_score_)

# With using these codes, we will be able to see ideal K value.
```

```
# Output:
Best Params: {‘n_neighbors’: 11}
Best Score: 0.975
```

3- Gradient Descent

Gradient descent is a kind of optimization algorithm which is used for minimizing the cost function in several ML algorithms. It is used for updating the parameters of learning model. There are several types of gradient descent.

• **Batch Gradient Descent:** All training examples of gradient descent be processed for each iteration. It's ideal to use when number of training examples aren't large since this kind of computations cost quite expensive.

• **Stochastic Gradient Descent:** Only one training example of gradient descent be processed for each iteration. Since parameters are being updated after each of iterations, its quite faster than Batch Gradient Descent. But still it's not ideal for cases which includes large number of training examples because it causes additional overhead for system.

• **Mini Batch Gradient Descent:** Batches of b training examples be processed in only one iteration even if number of training samples are large (where b<m). Therefore Mini Batch Gradient Descent algorithm works executes faster than both of other two algorithms.

```python
# Basic Gradient Descent Algorithm In Python [6]
import numpy as np

def gradient_descent(
    gradient, start, learn_rate, n_iter=50, tolerance=1e-06
):
    vector = start
    for _ in range(n_iter):
        diff = -learn_rate * gradient(vector)
        if np.all(np.abs(diff) <= tolerance):
            break
        vector += diff
    return vector

# Gradient: Function or callable object takes takes a input vector and
returns grandient of function which is trying to be minimized.
# Start: Point where algorithm start search, can be sequence or scalar.
# Learn_Rate: Learning rate which controls magnitude of vector update.
# N_Iter: Parameter of number of iterations.

# This function can be easily converted into another types of gradient
descent.
```

Also there are several optimization techniques for optimizing gradient descent too (SGD, Momentum, NAG etc.). :) These advanced methods are used for ensure finding a great solution which has higher probability of convergence.

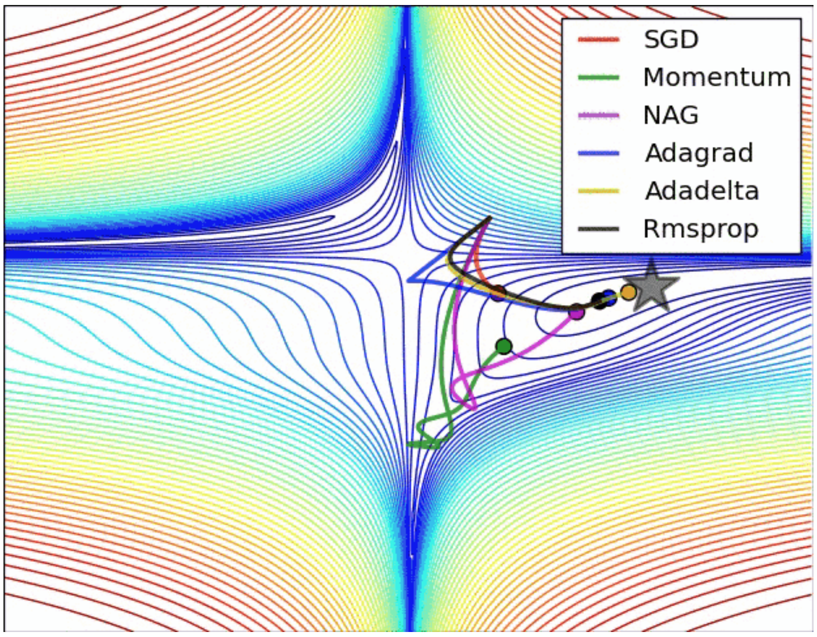


Figure-5: Different Methods for Optimizing Gradient Descent [7]

4- Genetic Algorithms

Genetic algorithms are search algorithms inspired by Darwin's Theory of Evolution in nature. [8] Genetic algorithms are frequently used in search, optimization and machine learning. Genetic algorithms can product quality solutions for various problems by simulating the processes of selection, reproduction and mutation. A genetic algorithm works with this logic:

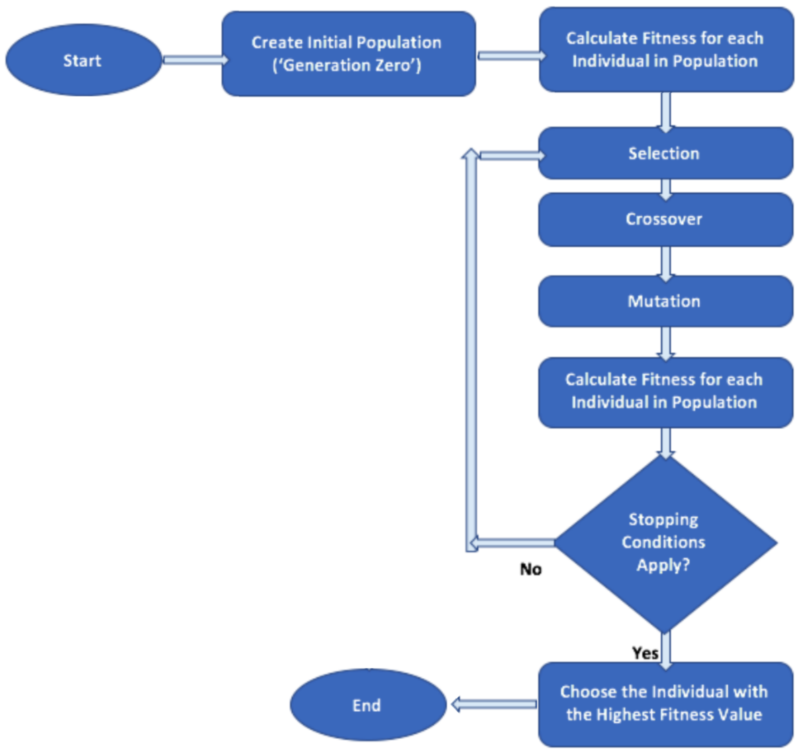

Start
Create Initial Population ('Generation Zero')
Calculate Fitness for each Individual in Population
Selection
Crossover
Mutation
Calculate Fitness for each Individual in Population
Stopping Conditions Apply?
No
Yes
Choose the Individual with the Highest Fitness Value
End

Figure-6: Basic Flow of a Genetic Algorithm

```
# Basic Genetic Algorithm In Python [9]
import pygad

y = f(w1:w6) = w1x1 + w2x2 + w3x3 + w4x4 + w5x5 + 6wx6
where (x1,x2,x3,x4,x5,x6)=(4,-2,3.5,5,-11,-4.7) and y=44
function_inputs = [4,-2,3.5,5,-11,-4.7]
desired_output = 44

def fitness_func(solution, solution_idx):
    output = numpy.sum(solution*function_inputs)
    fitness = 1.0 / numpy.abs(output - desired_output)
    return fitness

# Preparing parameters for pygad
fitness_function = fitness_func
num_generations = 50
num_parents_mating = 4
sol_per_pop = 8
num_genes = len(function_inputs)
init_range_low = -2
init_range_high = 5
parent_selection_type = 'sss'
keep_parents = 1
crossover_type = 'single_point'
mutation_type = 'random'
mutation_percent_genes = 10

ga_instance = pygad.GA(num_generations=num_generations,
                       num_parents_mating=num_parents_mating,
                       fitness_func=fitness_function,
                       sol_per_pop=sol_per_pop,
                       num_genes=num_genes,
                       init_range_low=init_range_low,
                       init_range_high=init_range_high,
                       parent_selection_type=parent_selection_type,
                       keep_parents=keep_parents,
                       crossover_type=crossover_type,
                       mutation_type=mutation_type,
                       mutation_percent_genes=mutation_percent_genes)

ga_instance.run()
solution, solution_fitness, solution_idx = ga_instance.best_solution()
print("Parameters of the best solution:
{solution}".format(solution=solution))
print("Fitness value of the best solution:
{solution_fitness}".format(solution_fitness=solution_fitness))

prediction = numpy.sum(numpy.array(function_inputs)*solution)
print("Predicted output based on the best solution :
{prediction}".format(prediction=prediction))
```

```
# Output:
Parameters of the best solution : [3.92692328 -0.11554946 2.39873381
3.29579039 -0.74091476 1.05468517]
Fitness value of the best solution = 157.37320042925006
Predicted output based on the best solution : 44.00635432206546
```

Algorithm	Advantages	Disadvantages	Where To Use
Exhaustive Search (Brute Force) Algorithms	• All possible options are evaluated. • The most intuitive one.	• When there are many solutions, it becomes extremely slow.	• The database is small. • High accuracy is more important than the cost and speed of comp.
Gradient Descent Algorithms	• Computation efficient. • Stable. • Easy and quick to use.	• Doesn't work if there are multiple local minima. • If the learning rate is too large, you risk skipping the right solution.	• Need to optimize the model fast. • You can’t calculate the parameters linearly and have to search for them.
Genetic Algorithms	• Can find good solutions in a short computation time. • Wide range of solutions (since it’s random).	• Can’t guarantee that the solution is optimal. • Hard to come up with good heuristics.	• Need to avoid getting stuck in local minima.

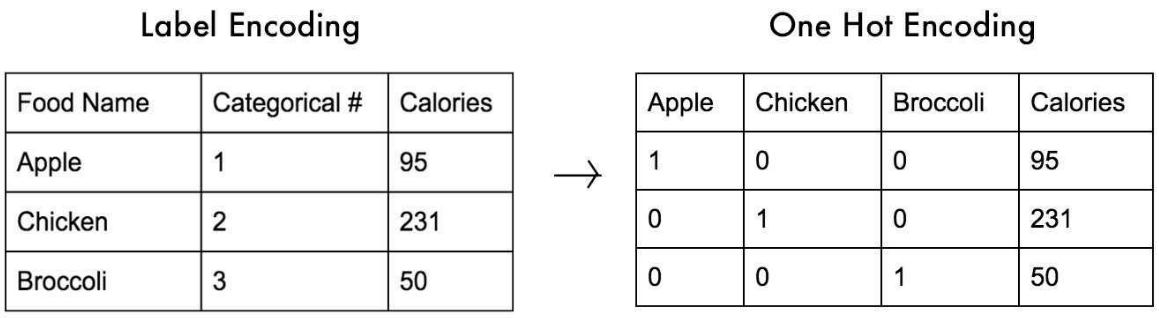
Figure-7: Differences Between ML Optimization Algorithms [3]

The advantages and disadvantages of each algorithm should be known and used in the right places and for the right purposes in order to achieve the highest efficiency. In addition, there are other things to consider in order for these algorithms to provide the highest benefit.

5- Another Things That Should Be Considered

✅ Missing data should be handled. There are several ways to do that such as dropping (variable, data entry etc.), replacing (by frequency, with average etc.) and leaving it as missing data. In order to boost performance of algorithm, if there are so much missing data they should be replaced (in order to prevent so much data loss).

✅ Machines (also ML algorithms) can’t understand text data. Therefore text data should be converted to integer (preferably binary) values. This is called as one-hot-encoding. With one-hot-encoding, machine will understand, analyze and perform data better. One hot encoding can be done easily with using “pd.get_dummies(df)” function of Pandas module in Python.



Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

Figure-8: Working Principle of One-Hot-Encoding [10]

✅ Underfitting and overfitting are problems that can be caused by wrong optimization procedures. Lots of training data causes true generalization error and less of training data causes underfitting problem. In order to prevent these situations, data should be splitted with distributing equally which is called cross validation. It’s more effective way to use of data.

Inputs
Overfitted Model
A
A
B
B
C
C
D E F
A B C
Expected Output
Model Output
Figure-9: Overfitting and Underfitting

✅ Model performance can be measured with using different methods and metrics. With using optimization techniques, performance of model can be improved and thus undesirable situations can be prevented.

While Developer Is Testing:
WORKING!
While Quality Control Team Is Testing:
WARNING!
While The Project Manager Is Testing:
404
While Users Are Testing:
Figure-10: Importance of Testing :) [11]

Simple Statistics About This Blog Post:

- 10 pages
- +1230 words (without codes and comment lines)
- +1700 words (with codes and comment lines)
- 5 different subsections
- 5 code blocks for 4 algorithms (with comments and sample outputs)
- 9 images
- 1 summary fable for comparing algorithms
- 11 references (First 9 are English and last 2 are Turkish)

References

1. https://www.researchgate.net/figure/Performance-of-four-ML-regression-algorithms-with-optimized-sets-of-hyperparameters-in_fig4_349103113
2. https://www.geeksforgeeks.org/ml-feature-scaling-part-2/
3. https://serokell.io/blog/ml-optimization
4. https://www.researchgate.net/figure/Brute-force-analytical-and-proposed-hybrid-machine-learning-algorithm_fig2_313454578
5. https://fda.readthedocs.io/en/latest/auto_examples/plot_k_neighbors_classification.html
6. https://realpython.com/gradient-descent-algorithm-python/
7. https://towardsdatascience.com/a-quick-overview-of-optimization-models-for-machine-learning-and-statistics-38e3a7d13138
8. https://www.analyticsvidhya.com/blog/2021/06/genetic-algorithms-and-its-use-cases-in-machine-learning/
9. https://pygad.readthedocs.io/en/latest/
10. https://womaneng.com/one-hot-encoding-nedir-nasil-yapilir/
11. https://technogezgin.com/debugging-nedir-nasil-yapilir/