# Calories Predicator SVM\_Grid (Support-Vector Machine)

# Introduction

Support-Vector Machine (SVM) are supervised learning models with associated learning algorithms, which analyze data used for classification and regression analysis. For our data set, we have done regression analysis to create the model.

The goal of the project is to predict the burnt calories given the 11 input features. The features included:

1. TotalSteps
2. TrackerDistance
3. LoggedActivitiesDistance
4. VeryActiveDistance
5. ModeratelyActiveDistance
6. LightActiveDistance
7. SedentaryActiveDistance
8. VeryActiveMinutes
9. FairlyActiveMinutes
10. LightlyActiveMinutes
11. SedentaryMinutes

We preprocess the data, then scale the features to range within , finally train and test SVM model.

# Model Implementation

## Data post-processing:

1. Load the data
2. Split the data into 80% train and 20% test sets
3. Scale the features

## Training:

1. Set up grid search
   1. Choose values for hyper-parameters. Models will be trained and test for the cross product of all hyper-parameter options.
   2. A 5-fold cross-validation was done for each set of hyper-parameters
   3. Choose the scoring function whose value is used to choose the best set of hyper-parameters
2. Initialize SVM model and perform grid search. An example is shown below

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.

[Parallel(n\_jobs=-1)]: Done 5 tasks | elapsed: 2.0s

[Parallel(n\_jobs=-1)]: Done 10 tasks | elapsed: 2.2s

[Parallel(n\_jobs=-1)]: Done 17 tasks | elapsed: 2.4s

[Parallel(n\_jobs=-1)]: Done 24 tasks | elapsed: 2.6s

[Parallel(n\_jobs=-1)]: Done 33 tasks | elapsed: 2.9s

[Parallel(n\_jobs=-1)]: Done 42 tasks | elapsed: 3.2s

[Parallel(n\_jobs=-1)]: Batch computation too fast (0.1939s.) Setting batch\_size=2.

[Parallel(n\_jobs=-1)]: Done 53 tasks | elapsed: 3.4s

[Parallel(n\_jobs=-1)]: Done 71 tasks | elapsed: 4.0s

[Parallel(n\_jobs=-1)]: Done 97 tasks | elapsed: 4.8s

[Parallel(n\_jobs=-1)]: Done 123 tasks | elapsed: 5.5s

[Parallel(n\_jobs=-1)]: Done 153 tasks | elapsed: 6.5s

[Parallel(n\_jobs=-1)]: Done 183 tasks | elapsed: 7.3s

[Parallel(n\_jobs=-1)]: Done 217 tasks | elapsed: 8.2s

[Parallel(n\_jobs=-1)]: Done 251 tasks | elapsed: 9.4s

[Parallel(n\_jobs=-1)]: Done 289 tasks | elapsed: 10.5s

[Parallel(n\_jobs=-1)]: Done 327 tasks | elapsed: 11.5s

[Parallel(n\_jobs=-1)]: Done 369 tasks | elapsed: 12.9s

[Parallel(n\_jobs=-1)]: Done 411 tasks | elapsed: 14.2s

[Parallel(n\_jobs=-1)]: Done 457 tasks | elapsed: 15.5s

[Parallel(n\_jobs=-1)]: Done 503 tasks | elapsed: 16.7s

[Parallel(n\_jobs=-1)]: Done 553 tasks | elapsed: 18.6s

[Parallel(n\_jobs=-1)]: Done 605 out of 605 | elapsed: 20.4s finished

The figure below plots r2 score vs different ‘C’ values used to train the SVM model. The solid green line indicates the mean r2 score of the SVM model trained on training set with 5-fold cross validation at each ‘C’ value. The dotted green line shows the same for models tested on testing set. As annotated on the figure, the best model is trained with a ‘C’ value of 1000 and achieves best r2 score of 0.71.

A close up of a map

Description generated with very high confidence

The following figure is similar to the figure above, expect that instead of r2 score it shows -RMSE value. The Root Mean Square (RMSE) value is a custom scoring function which is minimized to optimize the model.

A close up of a map

Description generated with high confidence

1. Select the set of hyper-parameters that were used to train the model that gave the best performance on the chosen scoring function. An example is given below:

Best parameters set found on development set:

{'C': 1000, 'epsilon': 100}

1. Train SVM model with this set of hyper-parameters using the entire training set
2. Test the trained SVM model on the testing set. The Root Mean Square Error (RMSE) and r2 score for the best GBR model is given below.

Training Root Mean Square Error (RMSE): 365.32555516174006

Testing Root Mean Square Error (RMSE): 348.90969608525774

Training R2 score: 0.7730868387558749

Testing R2 score: 0.7508698887661688

## Testing:

1. Load the test data or user given data
2. Scale the data using the params found during the training process
3. Load the model
4. Run the prediction

## Conclusion:

Training root mean square and testing root mean square values are very close and