Data Science Program

Capstone Report - Spring 2022

Machine Learning Analysis on Contextual and Covariate Features in Satellite Image of Lagos, Nigeria

Bradley Reardon – Individual Report

supervised by
Amir Jafari

Abstract

According to IDEAMAPS [1], at current, there is no systematic and scalable approach for mapping ‘Deprived’ areas throughout cities. With a map outlining which areas of a city are deprived, governments can more efficiently work towards building up the deprived areas without spending countless hours manually searching cities by foot. The approach this project took toward solving this issue is implementing deep learning and classical machine learning techniques on open-source data and free low resolution satellite imagery to classify and map areas as deprived or not. This approach proved to be an efficient and low-cost method for solving the issue at hand.

Table of Contents – Sections I Worked on and Wrote

1 Introduction

According to IDEAMAPS [1], at present, there is no systematic and scalable approach for mapping 'Deprived' areas throughout cities. Doing so is increasingly more difficult in low- and middle-income countries that do not have the funds, manpower, and infrastructure for efficient mapping.

A recent methodology for mapping the 'Deprived' areas involved satellite imagery and the use of machine learning algorithms to detect and classify areas as 'Deprived' or 'Built-up'. Because high quality satellite imagery is costly and difficult to access, the use of deep learning neural networks (particularly convolutional neural networks) often struggles with detecting deprived areas using low-resolution imagery. One potential solution to this problem while still implementing the use of machine learning algorithms is to apply traditional machine learning classification techniques on contextual and covariate feature data.

This project aimed to apply various deep learning and traditional machine learning classification techniques on low resolution and free satellite imagery as well as on calculated distance contextual and covariate features to detect deprived areas on a 10m^2 level.
The two core streams of the projects to detect deprived areas were as follows:

1 - Processing and usage of raw satellite images
2 - Utilizing covariate and contextual data

This report focused on the second stream to use labeled contextual and covariate features to train traditional classification techniques and classify deprived areas. The contextual and covariate feature data was mainly provided by open-source sources (OpenStreetMap and government data portals) and were computed using GIS software while the labeled satellite imagery was mainly provided by Ideamapsnetwork as ground observation work where deprived areas were mapped manually. In future efforts and cities, this will be conducted by teams local to said cities. The contextual features proved to be inefficient in providing enough information to accurately detect and classify deprived areas. However, the covariate features proved to be very useful in detecting which areas were considered ‘Deprived’.

2 Data Source

The data that was used to train and test the traditional machine learning models. There were two distinct datasets: contextual features and covariate features. Each consisting of labeled data comprised of three classes:

- Built-up (class 0)
- Deprived (class 1)
- Non-built-up (class 2)

2.1 Contextual Features

I helped work through the contextual features extraction and merging of coordinate data alongside Aziz and Jake.

Contextual features were defined as the statistical quantification of edge patterns, pixel groups, gaps, textures, and the raw spectral signatures calculated over groups of pixels or neighborhoods. [2]. These features can identify patterns and homogeneity in spatial configurations that go beyond spectral patterns or color intensities [2]. The contextual features dataset contains 144 distinct features of feature types: Fourier, Gabor, HOG, lacunarity, LBPM, LSR, mean, , normalized difference vegetation index (NDVI), ORB, PanTex, and SFS. The features were computed on a moving window size of 30-meters, 50-meters, and 70-meters which allows for capturing patterns at different scales, and zonal statistics – mean, sum, and standard deviation – were calculated on each contextual feature output [2].

The following quote from *Evaluating the Ability to Use Contextual Features Derived from Multi-Scale Satellite Imagery to Map Spatial Patterns of Urban Attributes and Population Distributions* [2] describes the contextual feature types in detail:

“ Fourier Transform. Fourier transform captures the frequency of patterns across an image. Any signal can be represented as a series of sinusoidal signals ; thus, an image can be decomposed into sine and cosine waves with various amplitudes and frequencies . The Fourier transform consists of magnitude and phase parts, with the former usually displayed as the output image (power spectrum). In these magnitude outputs, low-frequency features, such as water, are located closer towards the origin (center), with increasing frequency farther from the origin . A radial profile can be derived from a power spectrum, within which pixel frequencies can be summarized. Fourier produces two outputs: mean and variance.

Gabor. Gabor is a linear filter used for edge detection. Multiple filters consisting of strips are created by a sinusoidally modulated Gaussian function, forming the filter bank. The size, shape, and orientation of the filters can be set, and the various orientations enable extraction of features with those associated orientations. A Gabor wavelet transformation is outputted. There are 16 Gabor outputs: mean, variance, and 14 individual filters that examine different angles.
Histogram of Oriented Gradients. HOG identifies the orientation and magnitude of shades, distinguishing settlement and non-settlement classes. Gradient magnitudes in both the x and y

directions are calculated for each pixel and combined to obtain the magnitude and direction of the gradient. The image is divided into subregions (cells), and within each, the gradient direction bins the pixels by angles (1◦–180◦). The magnitude of each pixel is distributed to its associated bin, with the magnitude value split among two bins if the gradient direction falls between two. The aggregated magnitudes in each bin form a histogram (vector) for the cell.

Next, four cells (and their four histograms) are concatenated into a block and normalized. All block vectors are combined to form the final HOG vector, and statistics can be extracted. The five statistical outputs are the maximum, mean, variance, skew, and kurtosis.

Lacunarity. Lacunarity measures the homogeneity of the landscape via the spatial distribution of gap sizes. For heterogeneous images, all gap sizes are not the same; thus, the image is not translationally invariant, and lacunarity is high. For instance, in urban areas, there are gaps between buildings; in high density areas, there tend to be less gaps. Variation in gap sizes is scale dependent.

One way to calculate lacunarity involves a moving window in which the number of holes is calculated. First, an intensity surface, where the plane is the image and the z-axis (height) is the intensity (value) of the pixels, is created. A moving window of a set size is centered over one pixel, with a smaller gliding box placed in the upper left corner. If necessary, multiple boxes are stacked so all the pixel intensities fall within. The relative height is calculated using the minimum and maximum pixel values (or the boxes in which they fall) within the column. As the gliding box moves across the image window, all the relative heights are summed, and a formula is used to calculate lacunarity for that center pixel. The window repeats the process across the image. Only one lacunarity value is calculated.

Line Support Regions. LSR extracts straight lines from imagery, which can determine the area and spatial configuration of settled areas. Gradient orientations on an image are first calculated and used to group pixels into LSRs with similar gradient orientations. The groups that do not have enough support (pixels appropriated to a region, as described in Burns et al.) are removed. A plane fit to the pixel intensities in each line support region using a least squares fit and a horizontal plane of average pixel intensities, both weighted by local gradient magnitude, are created. A line is extracted where the two planes intersect. The line's length, width, contrast (intensity change over Remote Sens. 2021, 13, 3962 10 of 27 the line), steepness (slope of intensity change), and straightness can subsequently be obtained. LSR produces three outputs: line length, line mean, and line contrast.

Local Binary Pattern. LBPM assesses the homogeneity of an image, detecting bright and dark spots, flat areas, and edges. After the radius and number of neighbors are specified, the value of a center pixel is compared with those of its surrounding neighbors. If the center pixel value is smaller or equal, the neighbor is given a value of 1; otherwise, the value is 0. The values around the center pixel are taken sequentially (forming a binary string) and inputted into an equation to obtain the LBPM code for the center pixel. Patterns with more than two 0-1 or 1-0 switches are not uniform, with two or less considered uniform. A histogram is built with separate bins for each uniform pattern and one bin for all non-uniform patterns; this is based on Ojala et al.'s observation that certain uniform patterns appear more frequently in textures. Five statistical outputs of LBPM are produced: maximum, mean, variance, skew, and kurtosis.

Mean. The mean of the image is calculated using inverse distance weighting (IDW). IDW is an interpolation method where the influence of a point on an unknown point is inversely related with distance and dependent on the specified power setting, which controls the rate at which the influence of points decreases with increasing distance. For SpFeas, pixels near the center of a frame are given higher weights. In addition to mean, the variance of the pixels within the scale used is also calculated.

Normalized Difference Vegetation Index. NDVI assesses vegetation by incorporating a pixel’s value in the NIR and red regions. High values (towards 1) reflect a higher density of green vegetation, and low values (towards -1) reflect a lower density. NDVI values are generally lower in and negatively correlated with built-up areas due to sparser vegetation. Both the mean and variance of NDVI are calculated for each scale.

ORB. A feature-based matching method introduced by Rublee et al., ORB combines the Features from Accelerated Segment Test (FAST)—a feature detector—and Binary Robust Independent Elementary Features (BRIEF)—a feature descriptor—approaches.

The FAST algorithm is used to identify key points at each level in a scale pyramid of the image, and the Harris corner measure orders the key points and rejects edges picked up by FAS. Intensity centroid is used to assign an orientation to the corner. BRIEF selects a random pair of pixels around a key point, compares their intensity values, and assigns them binary values. The orientation from the intensity centroid is used to steer BRIEF towards this orientation, as BRIEF is not invariant to rotation. A greedy algorithm takes all the pairs and creates a subset (usually 256) of uncorrelated pairs, forming a 256-bit feature descriptor output (rotated BRIEF or rBRIEF). Five statistical outputs from ORB are produced: maximum, mean, variance, skew, and kurtosis.

PanTex. PanTex extracts built-up areas from panchromatic imagery using the GLCM approach. The textural contrast is calculated in all directions within a window around a pixel. The minimum value is taken, and the output with all the minimum values is the PanTex index. For urban areas, this minimum value would be consistently high. Pesaresi et al., used minimum values over average values, reasoning those averages produce an edge effect that could overestimate built-up areas. PanTex produces one output, which is the minimum contrast.

Structural Feature Sets. SFS extracts information on direction-lines. Lines from the center pixel are created in all directions. For a direction-line, a pixel is compared with the center pixel to determine whether it is considered homogenous. If it is, it is added to the direction line; the line keeps extending until a pixel is not considered homogenous based on set threshold levels or until the line reaches a set maximum length. This is repeated for all line directions. A histogram is built from the lines, and statistics can be extracted. SFS produces six outputs: maximum line length, minimum line length, mean, w-mean (weighted mean), standard deviation, and maximum ratio of orthogonal angles.” [2]

2.2 Covariate Features

The covariate features dataset consists of 61 distinct features from the following domains: Contamination, Facilities & services, Housing (HH) Housing (HO), Infrastructure, Physical hazards & assets, Population counts, SFS (HH), Social hazards & assets, Unplanned urbanization. These feature values were sourced from various open-source and government sources such as humdata.org, worldpop.org, geopode.org, spatialdatya.dhsprogram.com, and more. Please see the appendix for a table that lists each covariate feature, a short description of the feature, the data type, the domain the feature belongs to, and the link of the original data.

Covariate Feature Types	**Count**
Facilities & Services	8

Housing (HO)	1
Infrastructure	5
Physical Hazards & Assets	23
Population Counts	2
SES (HH)	14
Social Hazards & Assets	5
Unplanned Urbanization	3

(Figure 1: Covariate Feature Type Counts)

3 Problem Statement

There were two questions the researchers hoped to address in this study:

1. Can feature importance methods derive any contextual or covariate features that are useful in identifying ‘deprive’ and ‘built-up’ areas?

2. Through the use of classical machine learning models, are contextual and/or covariate features useful in identifying ‘Deprived’ and ‘Built-up’ areas

4 Related Work

Chao et al [2] found that analyzing contextual feature data from Sentinel-2 (10 m pixels) images in Accra, Belize, Ghana and Sri Lanka was useful in modeling population density and human modified landscape. The model these researchers developed yielded a coefficient of determination of 85% at very high spatial resolutions (<2m). With low resolution imaging (10 m) the model had an R^2 of 84%.

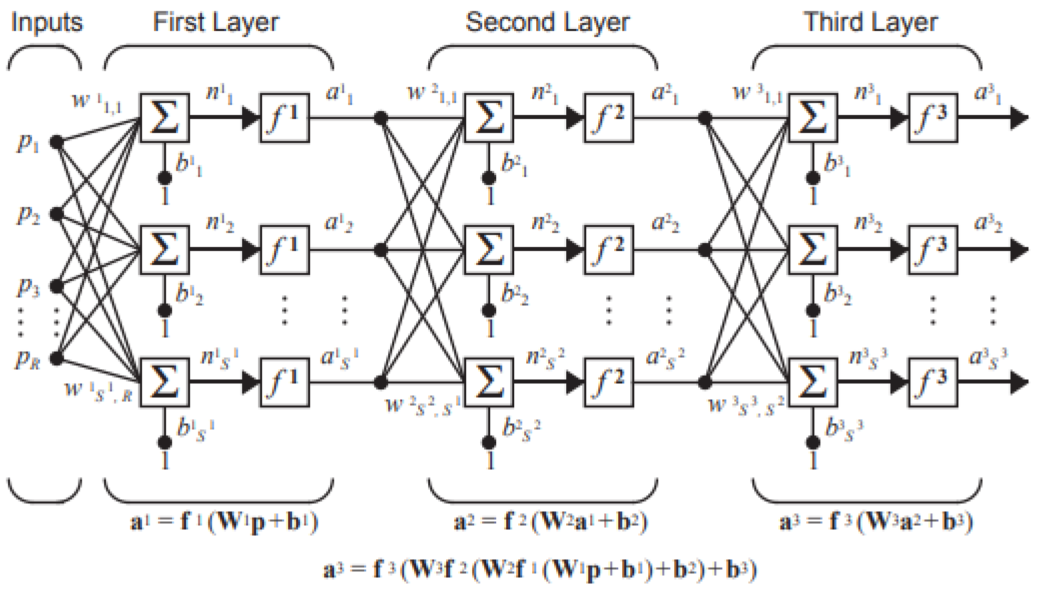
Saarela and Jauhiainen [3] utilized logistic regression with L1 penalization and non-linear (random forest) on the breast cancer data from UCI Archives and a running injury dataset. The goal of this research project was to see if combining feature importance techniques could provide more reliable results. The UCI breast cancer dataset contained 31 feature readings for breast masses identified (*smoothness, radius, symmetry*, …) as either malignant or benign. The different feature importance methods did have some overlap of features. The nine most important features for both methods had three overlapping values. The running injury dataset had a total of 85 features (*run level, left hip abductor, knee flexion peak of both legs*…) that described if someone was either injured or not. Random forest detected 22 important feature and logistic detected 61 important features. There was an overlap of only 13 features. This study demonstrated that introducing different feature importance methods can add to the robustness of the analysis. However, there may be limited overlap between different methods.

5 Solution and Methodology

5.5 Models

5.5.1 Multi-Layer Perceptron

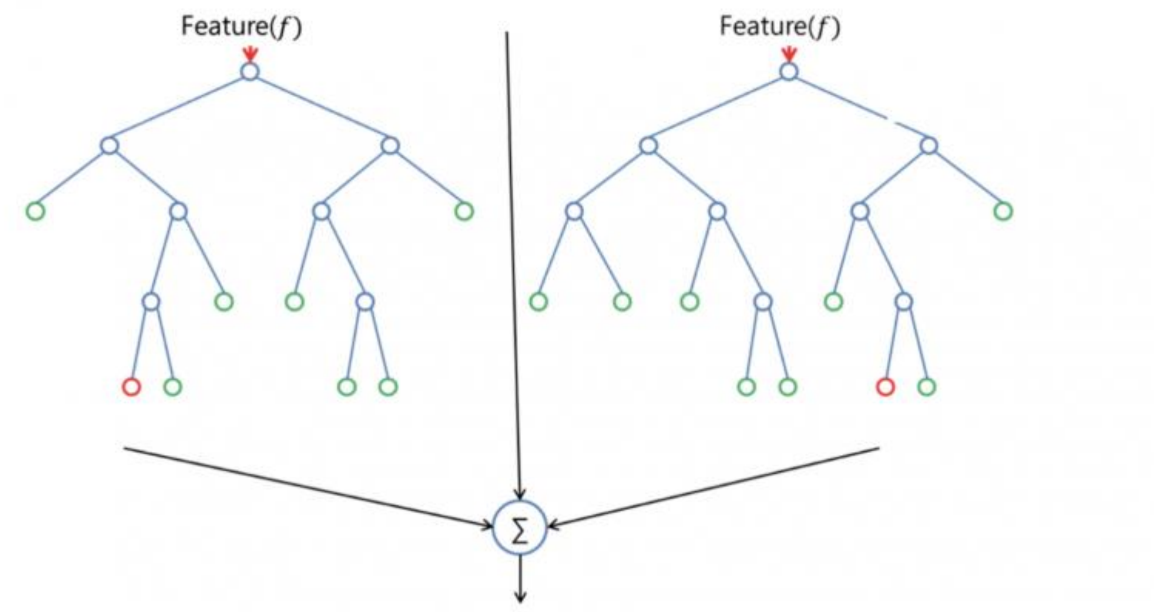
"A multilayer perceptron is a neural network connecting multiple layers in a directed graph, which means that the signal path through the nodes only goes one way. Each node, apart from the input nodes, has a nonlinear activation function. An MLP uses backpropagation as a supervised learning technique. Since there are multiple layers of neurons, MLP is a deep learning technique." [18]


Inputs
First Layer
Second Layer
Third Layer
$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)$
$\mathbf{a}^2 = \mathbf{f}^2(\mathbf{W}^2\mathbf{a}^1+\mathbf{b}^2)$
$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{a}^2+\mathbf{b}^3)$
$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$

(Figure 9: Multilayer Perceptron Architecture[19])

5.5.2 Random Forest

"Random forest is a supervised learning algorithm. The "forest" it builds is an ensemble of decision trees, usually trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result. Put simply: random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction." [20]



(Figure 10: Random Forest Containing Two Decision Trees [20])

A decision is made at each feature node per tree using calculated values (entropy, information gain) to determine which leaf (class) the feature node best represents.

5.5.3 Gradient Boosting

“Machine learning boosting is a method for creating an ensemble. It starts by fitting an initial model (e.g. a tree or linear regression) to the data. Then a second model is built that focuses on accurately predicting the cases where the first model performs poorly. The combination of these two models is expected to be better than either model alone. Then you repeat this process of boosting many times. Each successive model attempts to correct for the shortcomings of the combined boosted ensemble of all previous models.

Gradient boosting is a type of machine learning boosting. It relies on the intuition that the best possible next model, when combined with previous models, minimizes the overall prediction error. The key idea is to set the target outcomes for this next model in order to minimize the error. How are the targets calculated? The target outcome for each case in the data depends on how much changing that case's prediction impacts the overall prediction error:

- If a small change in the prediction for a case causes a large drop in error, then next target outcome of the case is a high value. Predictions from the new model that are close to its targets will reduce the error.

- If a small change in the prediction for a case causes no change in error, then next target outcome of the case is zero. Changing this prediction does not decrease the error.

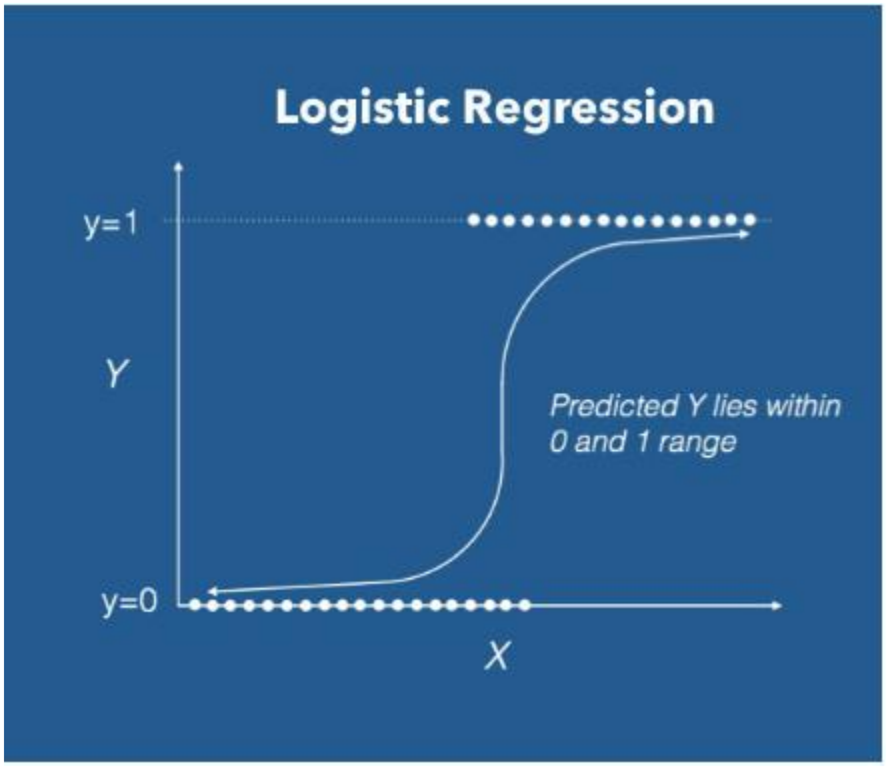
The name *gradient boosting* arises because target outcomes for each case are set based on the gradient of the error with respect to the prediction. Each new model takes a step in the direction that minimizes prediction error, in the space of possible predictions for each training case." [21]

5.5.4 Logistic Regression

"Logistic regression is a statistical analysis method to predict a binary outcome, such as yes or no, based on prior observations of a data set.

A logistic regression model predicts a dependent data variable by analyzing the relationship between one or more existing independent variables. For example, a logistic regression could be used to predict whether a political candidate will win or lose an election or whether a high school student will be admitted or not to a particular college. These binary outcomes allow straightforward decisions between two alternatives.

A logistic regression model can take into consideration multiple input criteria. In the case of college acceptance, the logistic function could consider factors such as the student's grade point average, SAT score and number of extracurricular activities. Based on historical data about earlier outcomes involving the same input criteria, it then scores new cases on their probability of falling into one of two outcome categories." [22]



(Figure 11: Logistic Regression Activation Function [23])

6 Results

6.1 Experimentation protocol

As mentioned in the *5 Solution and Methodology* section above, the model types we trained and tested on were: logistic regression, multi-layer perceptron (MLP) neural network, random forest, and gradient boosting classifiers. These models were chosen to give variety (linear, non-linear, & neural network models) in the model testing phase to see if any particular model type would best suit this classification problem.

The metric the researchers chose to identify as the indicator of model performance is the F1-score, particularly the micro F1-score for the deprived class and the macro F1-scrore for the entire model. The F1-score combines the precision and recall of a classifier into a single metric by taking their harmonic mean. It is primarily used to compare the performance of two classifiers and can be compared across models to determine which model performs best at correctly identifying and classifying instances [15]. The F1-score of a classification model is calculated as follows:

$$\frac{2(P * R)}{P + R}$$

P = the precision

R = the recall of the classification model

(Figure 50: F1-score Calculation)

The micro F1-score indicates how well a model was able to correctly classify the deprived instances since this class is very underrepresented, and the macro F1-score indicates how well a model was able to correctly classify all instances from every class.

Due to processing power and time limitations, the researchers limited the feature reduction tests to simply using the top 50 features in each feature set. The two feature importance methods used to create feature reduction sets were PCA and random forest feature importance. By iterating through all models, feature reduction sets, and included class combinations, we totaled 24 distinct models per covariate and contextual dataset, summing to 48 models total that were trained and tested.

To keep consistent across all experiments done using the Lagos satellite imagery since models across experiments were ensembled in an effort to maximize performance when collaborating with other researchers, this experiment primarily focused on the ability to correctly classify deprived areas using a subset of the contextual and covariate datasets only consisting of the deprived and built-up labeled classes.

For model optimization and tuning, we used Gridsearch CV [16] from the sklearn library and a hyper-parameter space for the MLP [17] and Logistic Regression [11] models. The standard base models for the Gradient Boosting [12] and Random Forest [9] models were used without any parameter tuning since we ran into unresolved grid search issues with those models.

The hyper-parameter spaces and parameters iterated through when looking for the optimal parameter combinations are as follows:

MLP

- Hidden layer sizes: (60, 100, 60), (100, 100, 100), (50, 100, 50)
- Activation: identity, relu, logistic, tanh
- Solver: adam
- Alpha: 0.0001
- Learning Rate: invscaling

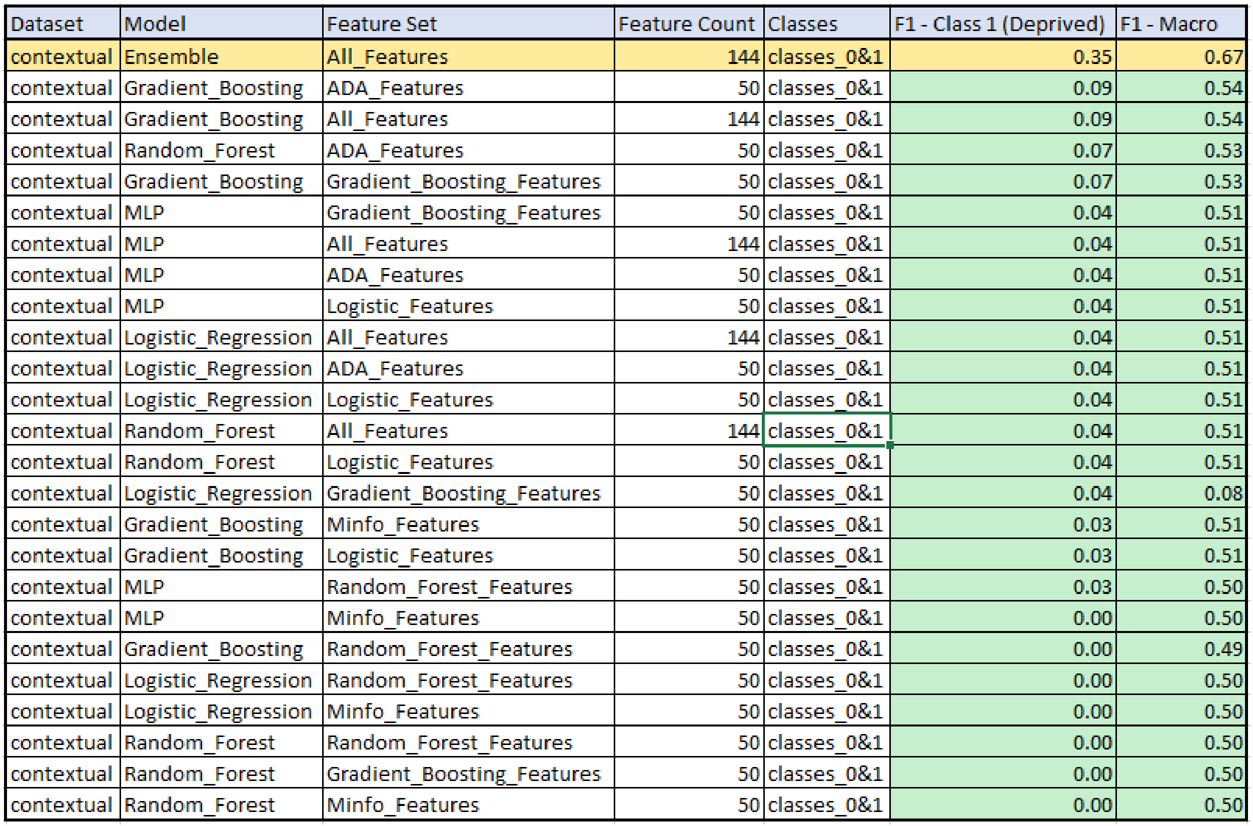
Logistic Regression

- Penalty: l1, l2, elasticnet, none
- Dual: True, False
- C: 0.001, 0.01, 1, 10
- Class Weight: dict, balanced, None
- Solver: newton-cg, lbfgs, liblinear, sag, saga

Due to processing power and time limitations, the researchers had to limit the number of parameter variables used in the grid search.

6.2 Contextual Features Results

Figure 51 compares the micro F1-scores for the deprived class and the macro F1-scores across all models trained and tested on a subset of contextual feature dataset including only classes 0 (Built-up) and 1 (Deprived).



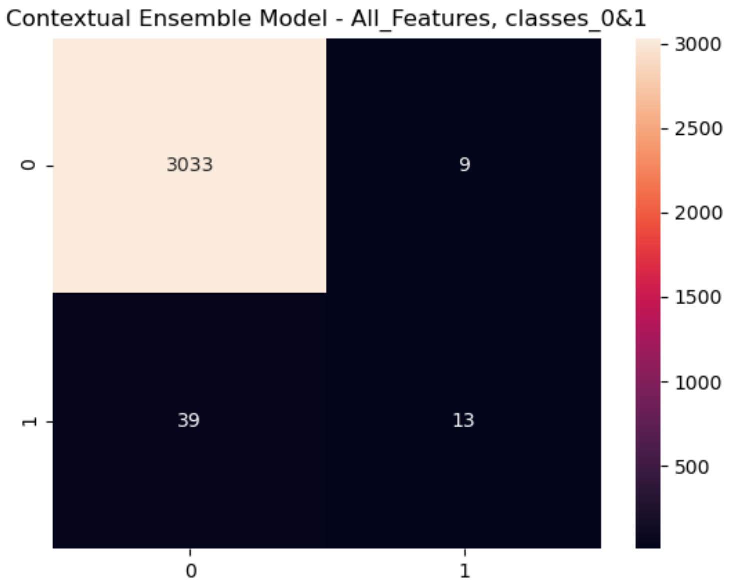
Dataset	Model	Feature Set	Feature Count	Classes	F1 - Class 1 (Deprived)	F1 - Macro
contextual	Ensemble	All_Features	144	classes_0&1	0.35	0.67
contextual	Gradient_Boosting	ADA_Features	50	classes_0&1	0.09	0.54
contextual	Gradient_Boosting	All_Features	144	classes_0&1	0.09	0.54
contextual	Random_Forest	ADA_Features	50	classes_0&1	0.07	0.53
contextual	Gradient_Boosting	Gradient_Boosting_Features	50	classes_0&1	0.07	0.53
contextual	MLP	Gradient_Boosting_Features	50	classes_0&1	0.04	0.51
contextual	MLP	All_Features	144	classes_0&1	0.04	0.51
contextual	MLP	ADA_Features	50	classes_0&1	0.04	0.51
contextual	MLP	Logistic_Features	50	classes_0&1	0.04	0.51
contextual	Logistic_Regression	All_Features	144	classes_0&1	0.04	0.51
contextual	Logistic_Regression	ADA_Features	50	classes_0&1	0.04	0.51
contextual	Logistic_Regression	Logistic_Features	50	classes_0&1	0.04	0.51
contextual	Random_Forest	All_Features	144	classes_0&1	0.04	0.51
contextual	Random_Forest	Logistic_Features	50	classes_0&1	0.04	0.51
contextual	Logistic_Regression	Gradient_Boosting_Features	50	classes_0&1	0.04	0.08
contextual	Gradient_Boosting	Minfo_Features	50	classes_0&1	0.03	0.51
contextual	Gradient_Boosting	Logistic_Features	50	classes_0&1	0.03	0.51
contextual	MLP	Random_Forest_Features	50	classes_0&1	0.03	0.50
contextual	MLP	Minfo_Features	50	classes_0&1	0.00	0.50
contextual	Gradient_Boosting	Random_Forest_Features	50	classes_0&1	0.00	0.49
contextual	Logistic_Regression	Random_Forest_Features	50	classes_0&1	0.00	0.50
contextual	Logistic_Regression	Minfo_Features	50	classes_0&1	0.00	0.50
contextual	Random_Forest	Random_Forest_Features	50	classes_0&1	0.00	0.50
contextual	Random_Forest	Gradient_Boosting_Features	50	classes_0&1	0.00	0.50
contextual	Random_Forest	Minfo_Features	50	classes_0&1	0.00	0.50

(Figure *51*: *Contextual Feature Model Results Using Only the Built-up (0) and Deprived (1) and Classes*)

The model that performed best on the subset of the contextual features dataset containing only instances for the Built-up (0) and Deprived (1) classes was a voting classifier ensemble model [24] model trained on the entire feature set. This ensemble model consisted of nine (9) MLP [18] models, all with the following parameters:

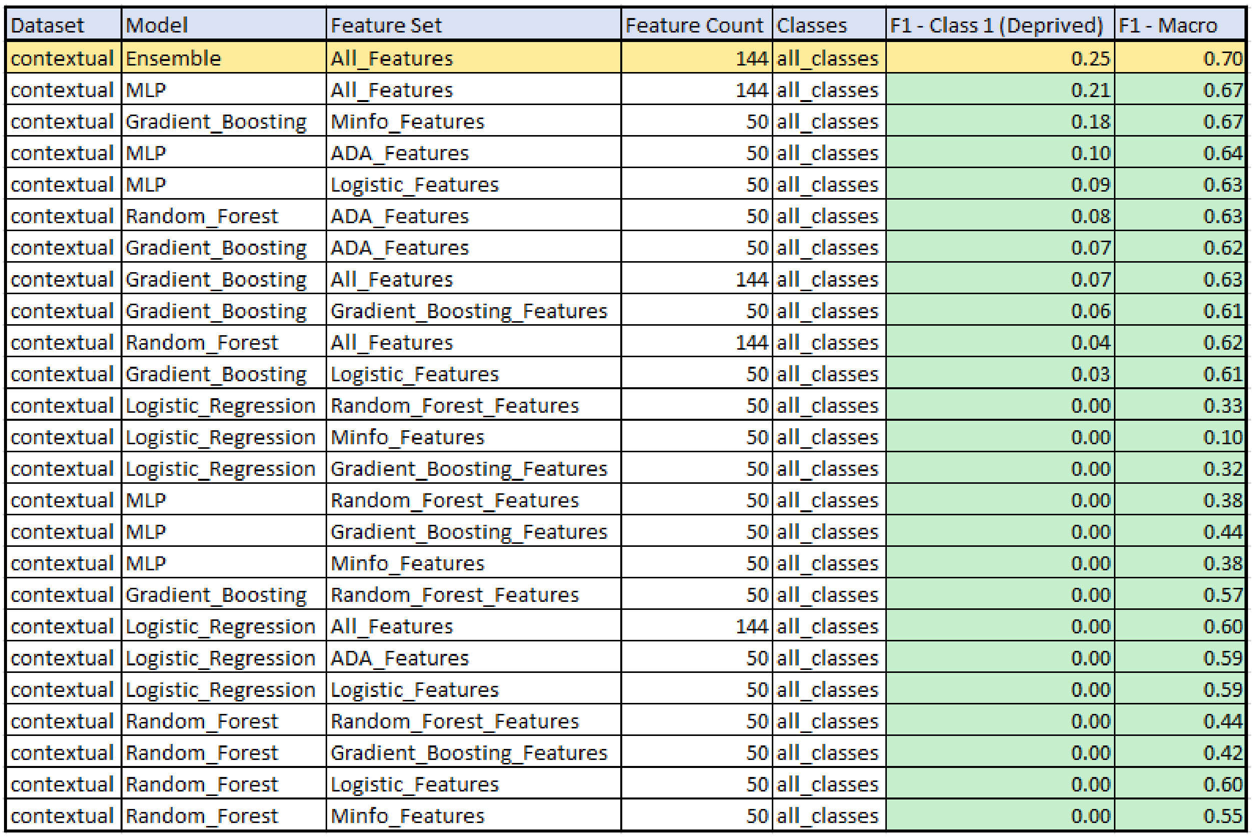
- Hidden layer sizes: (100, 100, 100)
- Activation: tanh
- Solver: adam
- Alpha: 0.0001
- Learning Rate: invscaling

These parameters were found to be the best parameters when using Gridsearch CV [16] on the independent MLP models and thus had the best performing parameters, meaning they would give the best performance for models used in the voting classification ensembled model.



(Figure 52: Confusion Matrix for the best performing model trained on the contextual features dataset containing only instances of the built-up (0) and deprived (1) classes.

Figure 53 compares the micro F1-scores for the deprived class and the macro F1-scores across all models trained and tested on the entire contextual features dataset, including all classes (built-up, deprived, and non-built-up).



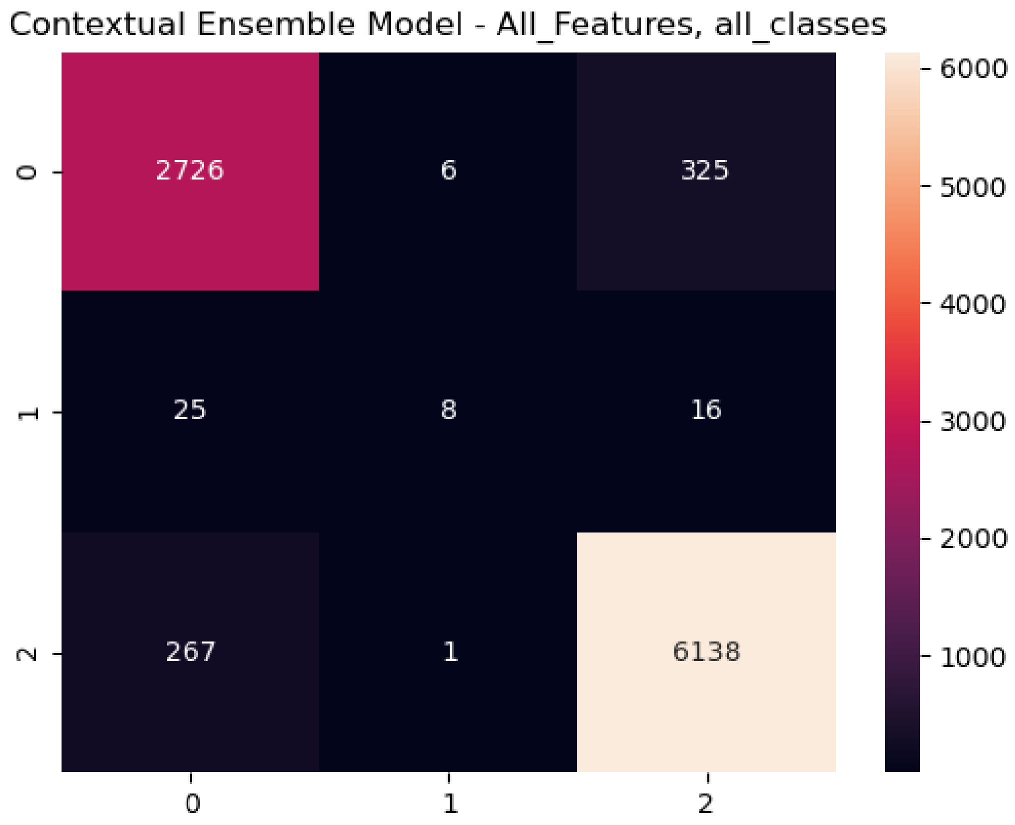
Dataset	Model	Feature Set	Feature Count	Classes	F1 - Class 1 (Deprived)	F1 - Macro
contextual	Ensemble	All_Features	144	all_classes	0.25	0.70
contextual	MLP	All_Features	144	all_classes	0.21	0.67
contextual	Gradient_Boosting	Minfo_Features	50	all_classes	0.18	0.67
contextual	MLP	ADA_Features	50	all_classes	0.10	0.64
contextual	MLP	Logistic_Features	50	all_classes	0.09	0.63
contextual	Random_Forest	ADA_Features	50	all_classes	0.08	0.63
contextual	Gradient_Boosting	ADA_Features	50	all_classes	0.07	0.62
contextual	Gradient_Boosting	All_Features	144	all_classes	0.07	0.63
contextual	Gradient_Boosting	Gradient_Boosting_Features	50	all_classes	0.06	0.61
contextual	Random_Forest	All_Features	144	all_classes	0.04	0.62
contextual	Gradient_Boosting	Logistic_Features	50	all_classes	0.03	0.61
contextual	Logistic_Regression	Random_Forest_Features	50	all_classes	0.00	0.33
contextual	Logistic_Regression	Minfo_Features	50	all_classes	0.00	0.10
contextual	Logistic_Regression	Gradient_Boosting_Features	50	all_classes	0.00	0.32
contextual	MLP	Random_Forest_Features	50	all_classes	0.00	0.38
contextual	MLP	Gradient_Boosting_Features	50	all_classes	0.00	0.44
contextual	MLP	Minfo_Features	50	all_classes	0.00	0.38
contextual	Gradient_Boosting	Random_Forest_Features	50	all_classes	0.00	0.57
contextual	Logistic_Regression	All_Features	144	all_classes	0.00	0.60
contextual	Logistic_Regression	ADA_Features	50	all_classes	0.00	0.59
contextual	Logistic_Regression	Logistic_Features	50	all_classes	0.00	0.59
contextual	Random_Forest	Random_Forest_Features	50	all_classes	0.00	0.44
contextual	Random_Forest	Gradient_Boosting_Features	50	all_classes	0.00	0.42
contextual	Random_Forest	Logistic_Features	50	all_classes	0.00	0.60
contextual	Random_Forest	Minfo_Features	50	all_classes	0.00	0.55

(Figure 53: Contextual Feature Model Results Using All Classes (Built-up, Deprived, Non-built-up)

The model that performed best on the subset of the contextual features dataset containing only instances for the Built-up (0) and Deprived (1) classes was a voting classifier ensemble model [24] model trained on the entire feature set. This ensemble model consisted of nine (9) MLP [18] models, all with the following parameters:

- Hidden layer sizes: (100, 100, 100)
- Activation: tanh
- Solver: adam
- Alpha: 0.0001
- Learning Rate: invscaling

These parameters were found to be the best parameters when using Gridsearch CV [16] on the independent MLP models and thus had the best performing parameters, meaning they would give the best performance for models used in the voting classification ensembled model.

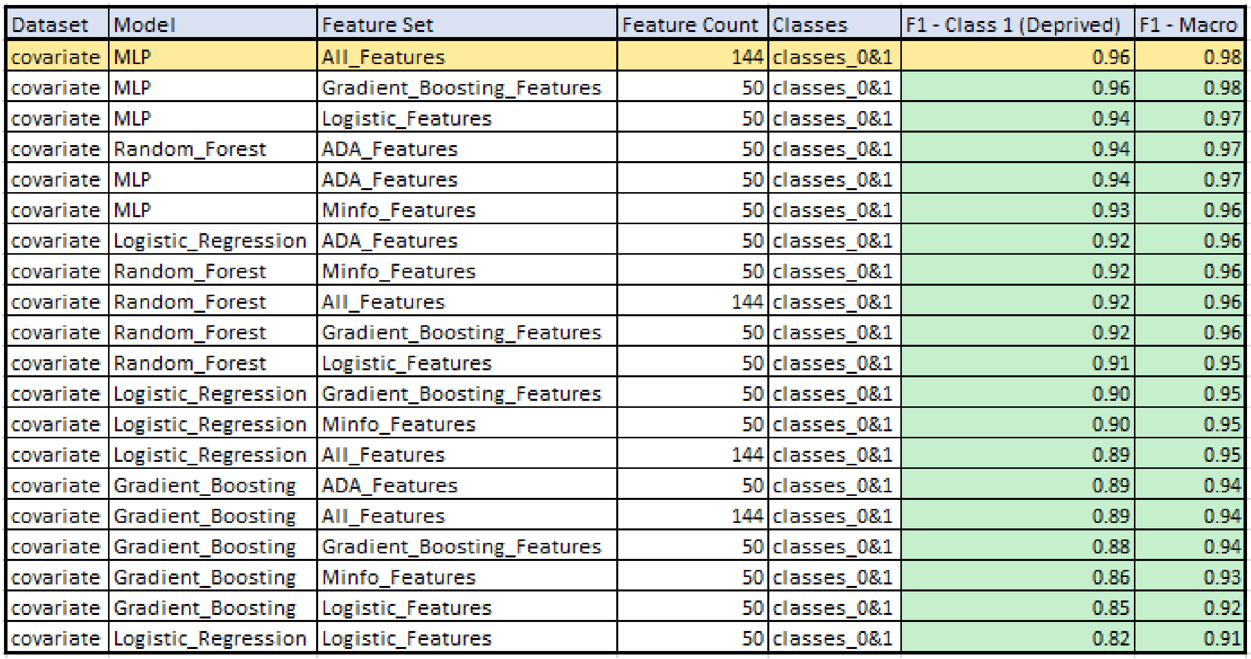


(Figure 54: Confusion Matrix for the best performing model trained on the full contextual features dataset containing all instances of the Built-up (0), Deprived (1), and Non-built-up (2) classes)

With the best model focused on the Built-up (0) and Deprived (1) classes having a micro F1-score for the Deprived classes of 0.35 and macro F1-score of 0.67, and the best model focused on all classes having a micro F1-score of 0.25 and a macro F1-score of 0.70, it is determined that the contextual features are not great indicators of class type.

6.3 Covariate Features Results

Figure 55 compares the micro F1-scores for the deprived class and the macro F1-scores across all models trained and tested on a subset of covariate features dataset including only classes 0 (Built-up) and 1 (Deprived).



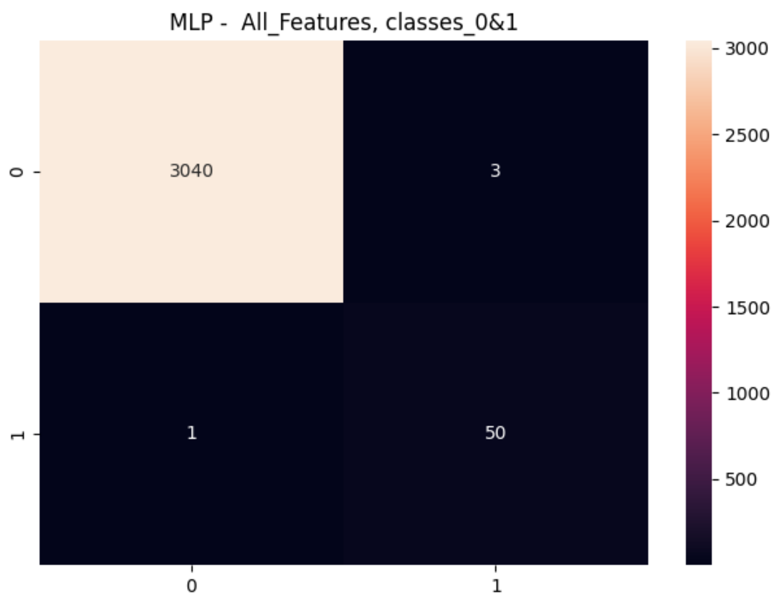
Dataset	Model	Feature Set	Feature Count	Classes	F1 - Class 1 (Deprived)	F1 - Macro
covariate	MLP	All_Features	144	classes_0&1	0.96	0.98
covariate	MLP	Gradient_Boosting_Features	50	classes_0&1	0.96	0.98
covariate	MLP	Logistic_Features	50	classes_0&1	0.94	0.97
covariate	Random_Forest	ADA_Features	50	classes_0&1	0.94	0.97
covariate	MLP	ADA_Features	50	classes_0&1	0.94	0.97
covariate	MLP	Minfo_Features	50	classes_0&1	0.93	0.96
covariate	Logistic_Regression	ADA_Features	50	classes_0&1	0.92	0.96
covariate	Random_Forest	Minfo_Features	50	classes_0&1	0.92	0.96
covariate	Random_Forest	All_Features	144	classes_0&1	0.92	0.96
covariate	Random_Forest	Gradient_Boosting_Features	50	classes_0&1	0.92	0.96
covariate	Random_Forest	Logistic_Features	50	classes_0&1	0.91	0.95
covariate	Logistic_Regression	Gradient_Boosting_Features	50	classes_0&1	0.90	0.95
covariate	Logistic_Regression	Minfo_Features	50	classes_0&1	0.90	0.95
covariate	Logistic_Regression	All_Features	144	classes_0&1	0.89	0.95
covariate	Gradient_Boosting	ADA_Features	50	classes_0&1	0.89	0.94
covariate	Gradient_Boosting	All_Features	144	classes_0&1	0.89	0.94
covariate	Gradient_Boosting	Gradient_Boosting_Features	50	classes_0&1	0.88	0.94
covariate	Gradient_Boosting	Minfo_Features	50	classes_0&1	0.86	0.93
covariate	Gradient_Boosting	Logistic_Features	50	classes_0&1	0.85	0.92
covariate	Logistic_Regression	Logistic_Features	50	classes_0&1	0.82	0.91

(Figure 55: Covariate Feature Model Results Using Only the Built-up (0) and Deprived (1) and Classes)

The model that performed best on the subset of the covariate features dataset containing only instances for the built-up (0) deprived (1) classes was an MLP model trained on the entire feature set. Through the use of Gridsearch CV and the parameter space mentioned in section 5.1, the following parameters were found to perform best for our top model highlighted above:

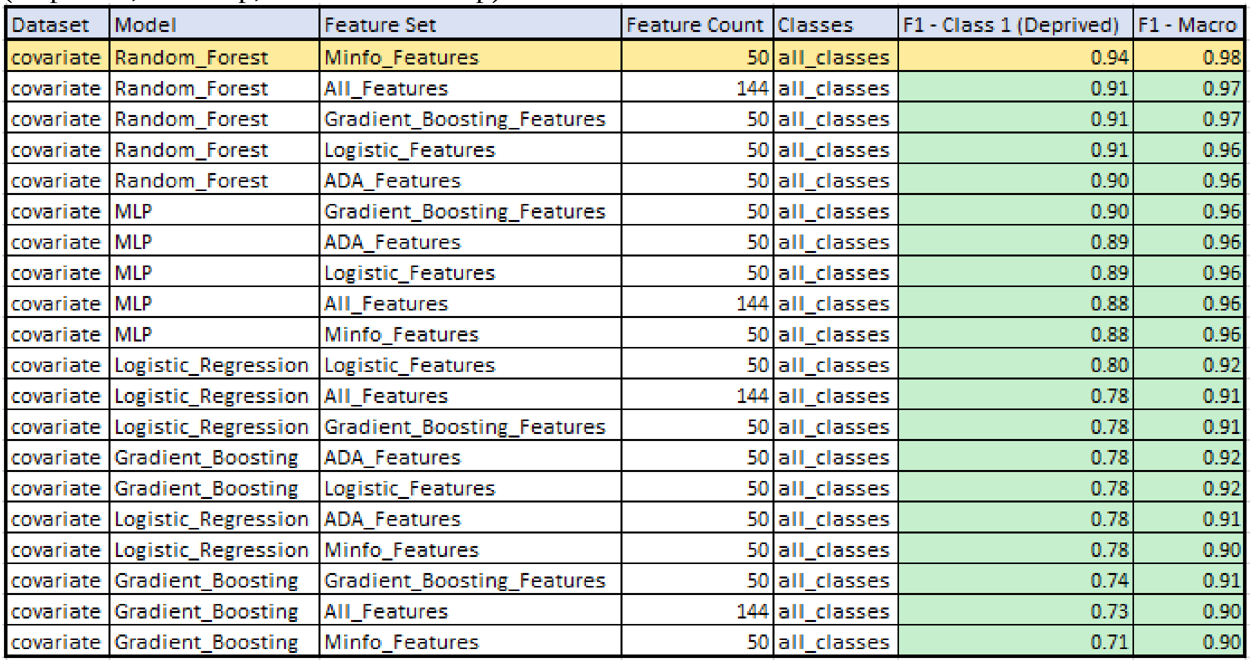
- Hidden layer sizes: (100, 100, 100)
- Activation: tanh
- Solver: adam
- Alpha: 0.0001
- Learning Rate: invscaling

This indicates that higher hidden layer sizes paired with the tanh activation outperforms lower hidden sizes and the other activations (identity, relu, logistic) for the dataset used to train and test these models.



(Figure 56 – Confusion Matrix for the best performing model trained on the covariate features dataset containing only instances of the Built-up (0) and Deprived (1) classes)

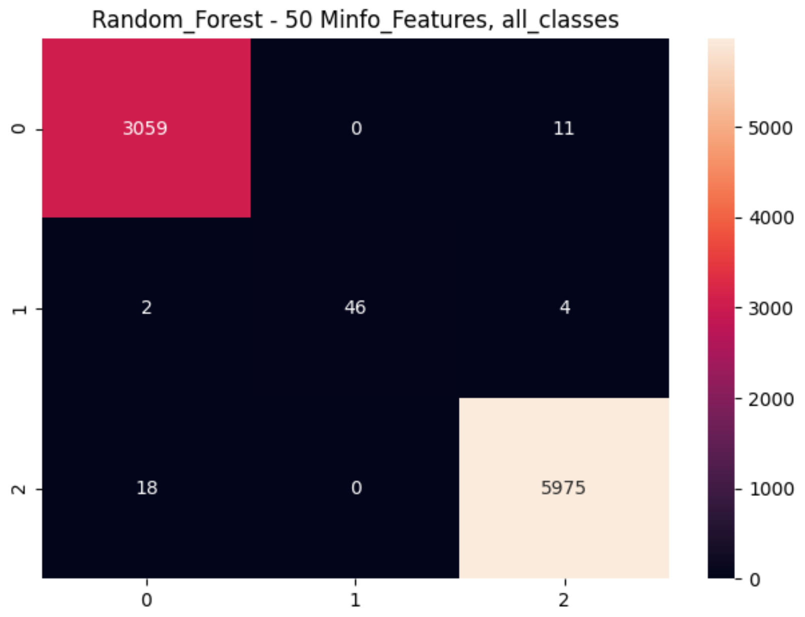
Figure 57 compares the micro F1-scores for the deprived class and the macro F1-scores across all models trained and tested on the entire covariate features dataset, including all classes (Deprived, Built-up, and Non-built-up).



Dataset	Model	Feature Set	Feature Count	Classes	F1 - Class 1 (Deprived)	F1 - Macro
covariate	Random_Forest	Minfo_Features	50	all_classes	0.94	0.98
covariate	Random_Forest	All_Features	144	all_classes	0.91	0.97
covariate	Random_Forest	Gradient_Boosting_Features	50	all_classes	0.91	0.97
covariate	Random_Forest	Logistic_Features	50	all_classes	0.91	0.96
covariate	Random_Forest	ADA_Features	50	all_classes	0.90	0.96
covariate	MLP	Gradient_Boosting_Features	50	all_classes	0.90	0.96
covariate	MLP	ADA_Features	50	all_classes	0.89	0.96
covariate	MLP	Logistic_Features	50	all_classes	0.89	0.96
covariate	MLP	All_Features	144	all_classes	0.88	0.96
covariate	MLP	Minfo_Features	50	all_classes	0.88	0.96
covariate	Logistic_Regression	Logistic_Features	50	all_classes	0.80	0.92
covariate	Logistic_Regression	All_Features	144	all_classes	0.78	0.91
covariate	Logistic_Regression	Gradient_Boosting_Features	50	all_classes	0.78	0.91
covariate	Gradient_Boosting	ADA_Features	50	all_classes	0.78	0.92
covariate	Gradient_Boosting	Logistic_Features	50	all_classes	0.78	0.92
covariate	Logistic_Regression	ADA_Features	50	all_classes	0.78	0.91
covariate	Logistic_Regression	Minfo_Features	50	all_classes	0.78	0.90
covariate	Gradient_Boosting	Gradient_Boosting_Features	50	all_classes	0.74	0.91
covariate	Gradient_Boosting	All_Features	144	all_classes	0.73	0.90
covariate	Gradient_Boosting	Minfo_Features	50	all_classes	0.71	0.90

(Figure 57: Covariate Feature Model Results Using All Classes (Built-up, Deprived, Non-built-up))

The two models that performed best on the full contextual features dataset containing all instances for every class were both Random Forest models, one trained on the entire feature set and the other trained on the top 50 features in the mutual information feature importance feature set.



(Figure 58: Confusion Matrix for the best performing model trained on the full covariate features dataset containing all instances of the Built-up (0), Deprived (1), and Non-built-up (2) classes)

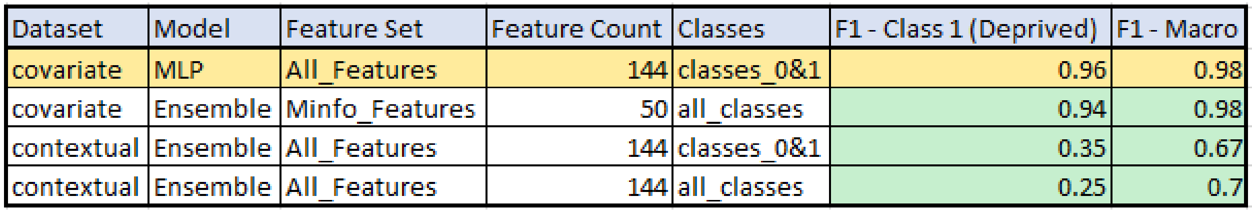
With the best model focused on the Built-up (0) and Deprived (1) classes having a micro F1-score for the Deprived classes of 0.96 and macro F1-score of 0.98, and the best model focused on all classes having a micro F1-score of 0.95 and a macro F1-score of 0.98, it is determined that the covariate features are extremely great indicators of class type..

7 Conclusion

This project as a whole aimed to apply various deep learning and traditional machine learning classification techniques on low resolution and free satellite imagery as well as on calculated contextual and covariate features to detect deprived areas on a 10m^2 level. This report specifically focused on the use of traditional machine learning classification techniques on the contextual and covariate features.

Two distinct datasets were tested in this experiment, the contextual features dataset and the covariate features dataset. Each test was split into two sub categories: testing on the full dataset including all classes (Built-up, Deprived, and Non-built-up) and testing on a subset of each dataset only consisting of the Built-Up and Deprived classes. The researchers focused on the dataset subsets in order to keep data consistent with researchers working on the deep-learning portion of the project.

Figure 59 represents the best models per dataset and subset of classes including the micro F1-score for the Deprived class and the macro F1-score for the entire model.



Dataset	Model	Feature Set	Feature Count	Classes	F1 - Class 1 (Deprived)	F1 - Macro
covariate	MLP	All_Features	144	classes_0&1	0.96	0.98
covariate	Ensemble	Minfo_Features	50	all_classes	0.94	0.98
contextual	Ensemble	All_Features	144	classes_0&1	0.35	0.67
contextual	Ensemble	All_Features	144	all_classes	0.25	0.7

(Figure 59: Best Models Results)

As you can see in the table, the covariate features are extremely great indicators of class type while the contextual features are not. It was found that in most cases, the full feature sets outperformed the top 50 features per feature importance set indicating that there is a loss of useful information when conducting feature reduction.

8 Bibliography

[1] Kabaria, Carloine Wanjiku, "Integrated Deprived Area Mapping System (IDEAMAPS) Newtwork", *UK Research and Innovation,* 2021 https://gtr.ukri.org/project/0D56FCFB-5673-4AF2-8091-0A1710B6C3F4
[2] Chao, Steven, "Evaluating the Ability to Use Contextual Features Derived from Multi-Scale Satellite Imagery to Map Spatial Patterns of Urban Attributes and Population Distributions", *Remote Sensing*, 2021
[3] Jauhianinen, Saarela, 'Comparison of feature importance measures as explanations for classification models', https://link.springer.com/article/10.1007/s42452-021-04148-9
[4] sci-kit learn, StandardScalar, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
[5] sci-kit learn, SelectKbest, https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
[6] sci-kit learn, mutual_info_classif, https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html
[7] Grassberger, Stögbauer, and Kraskov, 'Estimating mutal information, https://journals.aps.org/pre/abstract/10.1103/PhysRevE.69.066138
[8] Aznar, 'What is Mutual Information', https://quantdare.com/what-is-mutual-information/
[9]sci-kit learn, *Random Forest Classifie*r, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[10] sci-kit learn, 'Feature Importance with a Forest of Trees', https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html
[11] sci-kit learn, *Logistic Regression Classifie*r, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[12] sci-kit learn, *Gradient Boosting Classifier*, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
[13] sci-kit learn, *AdaBoost Classifier*, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#sklearn.ensemble.AdaBoostClassifier.feature_importances_

[14] sci-kit learn, *Stratified-KFold*, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html
[15] Edpresso Team, "What is the F1-score?", *Educative*, https://www.educative.io/edpresso/what-is-the-f1-score
[16] sci-kit learn, *Gridsearch CV*, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[17] sci-kit learn, *MLPClassifier*, *https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html*
[18] Techopedia, *Multi-Layer Perceptron (MLP)*, https://www.techopedia.com/definition/20879/multilayer-perceptron-mlp#:~:text=A%20multilayer%20perceptron%20is%20a,as%20a%20supervised%20learning%20technique.
[19] Hagan, Martin T., *Neural Network Design 2nd Edition*, https://hagan.okstate.edu/NNDesign.pdf
[20] Donges, Niklas, *Random Forest Algorithm: A Complete Guide*, https://builtin.com/data-science/random-forest-algorithm#how
[21] Hoare, Jake, "Gradient Boosting Explained – The Coolest Kid on The Machine Learning Block", *DisplayR*, https://www.displayr.com/gradient-boosting-the-coolest-kid-on-the-machine-learning-block/#:~:text=Gradient%20boosting%20is%20a%20type,order%20to%20minimize%20the%20error.
[22] Lawton, George, "Logistic Regression", *TechTarget*, https://www.techtarget.com/searchbusinessanalytics/definition/logistic-regression#:~:text=Logistic%20regression%20is%20a%20statistical,or%20more%20existing%20independent%20variables.
[23] Varsheni, Shri, "perform Logistic Regression with Pytorch Seamlessly", *Analytics Vidya*, https://www.analyticsvidhya.com/blog/2021/07/perform-logistic-regression-with-pytorch-seamlessly/, *2021*
[24] sci-kit learn, *Voting Classifier*, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html

9 Appendix

Run_Models.py

```
# Script for running the run_model function

import pandas as pd
import os.path
```

```
from Models import run_model

from project_root import get_project_root
root = get_project_root()



# all potential parameters for model testing
datasets = ['contextual', 'covariate']
models = ['MLP', 'Gradient_Boosting', 'Logistic_Regression', 'Random_Forest']
features = ['All_Features', 'ADA_Features', 'Random_Forest_Features',
'Gradient_Boosting_Features', 'Logistic_Features', 'Minfo_Features']
# feature_count = 2 #Any integer. Not required if using full feature set(use
'' to remove count from titles)
classes = ['all_classes', 'classes_0&1']

# desired model parameter combinations
datasets = ['contextual']
models = ['MLP', 'Gradient_Boosting', 'Logistic_Regression', 'Random_Forest']
features = ['All_Features', 'ADA_Features', 'Random_Forest_Features',
'Gradient_Boosting_Features', 'Logistic_Features', 'Minfo_Features']
classes = ['all_classes']

dataset_list = []
model_list = []
features_list = []
feat_count_list = []
classes_list = []
deprived_list = []
macro_list = []


# iterate through desired models
for dataset in datasets:
    for model in models:
        for feat in features:
            if feat == 'All_Features' and dataset == 'contextual':
                feature_count = 144
            elif feat == 'All_Features' and dataset == 'covariate':
                feature_count = 60
            else:
                feature_count = 50
            for class_count in classes:
                stored_model_info = run_model(dataset=dataset,
                                              model=model,
                                              features=feat,
                                              feature_count=feature_count,
                                              classes=class_count)
                stored_model_info # prints classification report table and
confusion matrix image
                dataset_list.append(dataset) # dataset used
                model_list.append(model) # model name
                features_list.append(feat) # feature set used
                feat_count_list.append(feature_count) # number of features
trained on
                classes_list.append(class_count) # classes trained and tested
on
```

```
                deprived_list.append(stored_model_info[1]) # f1 score for
class 1 (deprived)
                macro_list.append(stored_model_info[2]) # macro f1 score


ensemble_results = run_model(dataset='contextual',
                             model='Ensemble',
                             features='All_Features',
                             feature_count=144,
                             classes='all_classes')
dataset='contextual'
model='Ensemble'
features='All_Features'
feature_count=144
classes='all_classes'

# store ensemble data into table
ensemble_results # prints classification report table and confusion matrix
image
dataset_list.append(dataset) # dataset used
model_list.append(model) # model name
features_list.append(features) # feature set used
feat_count_list.append(feature_count) # number of features trained on
classes_list.append(classes) # classes trained and tested on
deprived_list.append(ensemble_results[1]) # f1 score for class 1 (deprived)
macro_list.append(ensemble_results[2]) # macro f1 score


# store data in dataframe
data = {'Dataset': dataset_list,
        'Model': model_list,
        'Feature Set': features_list,
        'Feature Count': feat_count_list,
        'Classes': classes_list,
        'F1 - Class 1 (Deprived)': deprived_list,
        'F1 - Macro': macro_list}
comparison_table = pd.DataFrame(data)
print(comparison_table)

# write dataframe to csv
filename = 'Model_Results_Comparison_Table.csv'
comparison_table.to_csv(root / '1.Data' / f'{filename}', index=False)
```

Models.py

```
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix, f1_score
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
import json
import os
import warnings
from project_root import get_project_root
warnings.filterwarnings("ignore")

root = get_project_root()

def run_model(dataset ='', model='', features='', feature_count=50, classes =
''):

    if dataset == 'contextual':
        # Read in dataframe and remove merged columns
        df = pd.read_csv(root / '1.Data' / 'Contextual_Features_final.csv')
        df = df.drop(columns=['long', 'lat', 'Point'])
        cols_to_move = ['Label']
        df = df[cols_to_move + [col for col in df.columns if col not in
cols_to_move]]

        # Move Target to first column
        target = 'Label'
        first_col = df.pop(target)
        df.insert(0, target, first_col)

    elif dataset == 'covariate':
        # Read in dataframe and remove merged columns
        df = pd.read_csv(root / '1.Data' / 'Covariate_Features.csv')
        df.drop(['long', 'lat'], axis=1, inplace=True)
        df.dropna(inplace=True)
        cols_to_move = ['Label']
        df = df[cols_to_move + [col for col in df.columns if col not in
cols_to_move]]

        # Move Target to first column
        target = 'Label'
        first_col = df.pop(target)
        df.insert(0, target, first_col)

    if classes == 'all_classes':
        df = df
    elif classes == 'classes_0&1':
        # set label column equal to only 0 and 1 classes
        df = df[df['Label'].isin([0, 1])]

    # define target and independent features
    if features == 'All_Features': # full dataset, all features

        feature_count = ''
        X = df.values[:, 1:]
        y = df.values[:, 0]

    elif features == 'ADA_Features': # PCA feature selection

        if dataset == 'contextual':
            filename = 'Contextual_best_ada_boosting_features_0_1.csv'
            ada_features = pd.read_csv(root /
'3.Contextual_and_Covariate_Feautres_Modeling'/ 'feature_selection' /
'Contextual' / f'{filename}', index=False)
        else:
            filename = 'Covariate_best_ada_boosting_features_0_1.csv'
            ada_features = pd.read_csv(root /
'3.Contextual_and_Covariate_Feautres_Modeling' / 'feature_selection' /
'Covariate' / f'{filename}',index=False)

        ada = ['Label']
        for row in range(feature_count):
            ada.append(ada_features.iloc[row, 0])

        df_ada = df[ada]
        X = df_ada.values[:, 1:]
        y = df_ada.values[:, 0]

    elif features == 'Random_Forest_Features':  # Random Forest feature
selection

        if dataset == 'contextual':
            filename = 'Contextual_best_random_forest_features_0_1.csv'
            rf_features = pd.read_csv(root /
'3.Contextual_and_Covariate_Feautres_Modeling' / 'feature_selection' /
'Contextual' / f'{filename}',index=False)
        else:
            filename = 'Covariate_best_random_forest_features_0_1.csv'
            rf_features = pd.read_csv(root /
'3.Contextual_and_Covariate_Feautres_Modeling' / 'feature_selection' /
'Covariate' / f'{filename}',index=False)

        rf = ['Label']
        for row in range(feature_count):
            rf.append(rf_features.iloc[row,0])

        df_rf = df[rf]

        X = df_rf.values[:, 1:]
        y = df_rf.values[:, 0]

    elif features == 'Gradient_Boosting_Features':  # Random Forest feature
selection

        if dataset == 'contextual':
            filename = 'Contextual_best_gradient_boosting_features_0_1.csv'
            gb_features = pd.read_csv(root /
'3.Contextual_and_Covariate_Feautres_Modeling' / 'feature_selection' /
'Contextual' / f'{filename}',index=False)
        else:
            filename = 'Covariate_best_gradient_boosting_features_0_1.csv'
            gb_features = pd.read_csv(root /
'3.Contextual_and_Covariate_Feautres_Modeling' / 'feature_selection' /
'Covariate' / f'{filename}',index=False)

        gb = ['Label']
        for row in range(feature_count):
            gb.append(gb_features.iloc[row,0])
        df_gb = df[gb]

        X = df_gb.values[:, 1:]
        y = df_gb.values[:, 0]

    elif features == 'Logistic_Features':  # Random Forest feature selection

        if dataset == 'contextual':
            filename = 'Contextual_best_logistic_features_features_0_1.csv'
            log_features = pd.read_csv(root /
'3.Contextual_and_Covariate_Feautres_Modeling' / 'feature_selection' /
'Contextual' / f'{filename}',index=False)
        else:
            filename = 'Covariate_best_logistic_features_features_0_1.csv'
            log_features = pd.read_csv(root /
'3.Contextual_and_Covariate_Feautres_Modeling' / 'feature_selection' /
'Covariate' / f'{filename}',index=False)

        log = ['Label']
        for row in range(feature_count):
            log.append(log_features.iloc[row,0])

        df_log = df[log]

        X = df_log.values[:, 1:]
        y = df_log.values[:, 0]

    elif features == 'Minfo_Features':  # Random Forest feature selection

        if dataset == 'contextual':
            filename = 'Contextual_best_minfo_features_features_0_1.csv'
            minfo_features = pd.read_csv(root /
'3.Contextual_and_Covariate_Feautres_Modeling' / 'feature_selection' /
'Contextual' / f'{filename}',index=False)
        else:
            filename = 'Covariate_best_minfo_features_features_0_1.csv'
            minfo_features = pd.read_csv(root /
'3.Contextual_and_Covariate_Feautres_Modeling' / 'feature_selection' /
'Covariate' / f'{filename}',index=False)

        minfo = ['Label']
        for row in range(feature_count):
            minfo.append(minfo_features.iloc[row,0])

        df_minfo = df[minfo]

        X = df_minfo.values[:, 1:]
        y = df_minfo.values[:, 0]


    # train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
    X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.25,
                                                      random_state=42)  #
0.25 x 0.8 = 0.2
```

```
    # Feature Scaling
    sc = StandardScaler()
    sc.fit(X_train)
    X_train = sc.transform(X_train)
    X_test = sc.transform(X_test)
    X_val = sc.transform(X_val)


    # Model file saved and exists externally

    # directory path models are saved in
    # directory =
r'3.Contextual_and_Covariate_Feautres_Modeling/Saved_Models'
    directory = r'C:\Users\brear\OneDrive\Desktop\Grad School\Mapping-
Deprived-Areas-Using-Deep-Neural-
Networks\3.Contextual_and_Covariate_Feautres_Modeling\Saved_Models'
    # model filename
    filename =
f'{dataset}_{model}_model_{feature_count}{features}_{classes}.sav'

    if not os.path.exists(f'{directory}/{filename}'):
        if model == 'MLP':
            # Hyper-parameter space
            '''
            parameter_space = {
                'hidden_layer_sizes': [(60, 100, 60), (100, 100, 100), (50,
100, 50)],
                'activation': ['identity', 'relu', 'logistic', 'tanh'],
                'solver': ['sgd', 'adam', 'lbfgs'],
                'alpha': [0.0001, 0.00001, 0.000001],
                'learning_rate': ['constant', 'adaptive', 'invscaling'],
            }
            '''

            parameter_space = {
                'hidden_layer_sizes': [(60, 100, 60), (100, 100, 100), (50,
100, 50)],
                'activation': ['identity', 'relu', 'logistic', 'tanh'],
                'solver': ['adam'],
                'alpha': [0.0001],
                'learning_rate': ['invscaling'],
            }

            # Create network
            clf = MLPClassifier(max_iter=1000000)

            # Run Gridsearch
            clf = GridSearchCV(clf, parameter_space, n_jobs=-1, cv=3)

            clf.fit(X_train, y_train)
            clf_pred = clf.predict(X_test)
            print(f"Test Results Using {dataset} dataset {model} Best Params,
{feature_count}{features}, and {classes}: \n")
            print("Classification Report: ")
            print(classification_report(y_test, clf_pred))
            # Best parameter set
            print(f'Best parameters found for {model}:\n', clf.best_params_)

            # Save model
            filename =
f'{dataset}_{model}_model_{feature_count}{features}_{classes}.sav'
            pickle.dump(clf, open(f'{directory}/{filename}', 'wb'))

        elif model == "Gradient_Boosting":
            # Hyper-parameter space
            '''
            parameter_space = {
                'loss': ['deviance'],
                'criterion': ['friedman_mse', 'squared_error', 'mse'],
                'n_estimators': [100, 200, 50],
                'subsample': [1.0, 0.8, 0.6],
                "learning_rate": [0.01, 0.025, 0.05],
                "min_samples_split": np.linspace(0.1, 0.5, 12),
                "min_samples_leaf": np.linspace(0.1, 0.5, 12),
                "max_depth": [3, 5, 8],
                "max_features": ["log2", "sqrt"],
            }


            parameter_space = {
                'loss': ['deviance'],
                'criterion': ['friedman_mse', 'mse'],
                'n_estimators': [100],
                'subsample': [1.0, 0.6],
                "learning_rate": [0.01, 0.05],
                "min_samples_split": np.linspace(0.1, 0.5, 3),
                "min_samples_leaf": np.linspace(0.1, 0.5, 3),
                "max_depth": [3, 8],
                "max_features": ["log2", "sqrt"],
            }
            '''
            clf = GradientBoostingClassifier()

            # Run Gridsearch
            # clf = GridSearchCV(clf, parameter_space, n_jobs=-1, cv=3)

            clf.fit(X_train, y_train)
            clf_pred = clf.predict(X_test)
            print(f"Test Results Using {model}, {feature_count}{features},
and {classes}: \n")
            print("Classification Report: ")
            print(classification_report(y_test, clf_pred))

            # Best parameter set
            # print(f'Best parameters found for {model}:\n',
clf.best_params_)

            # Save model
            filename =
f'{dataset}_{model}_model_{feature_count}{features}_{classes}.sav'
            pickle.dump(clf, open(f'{directory}/{filename}', 'wb'))
        elif model == "Logistic_Regression":
            # Logistic Regression Hyper-parameter space
            parameter_space = {
                'penalty': ['l1', 'l2','elasticnet', 'none'],
                'dual': [True, False],
                'C': [0.001, 0.01, 0.1, 1, 10],
                'class_weight': ['dict', 'balanced', None],
                'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
            }

            clf = LogisticRegression()

            # Run Gridsearch
            clf = GridSearchCV(clf, parameter_space, n_jobs=-1, cv=3)

            clf.fit(X_train, y_train)
            clf_pred = clf.predict(X_test)
            print(f"Test Results Using {dataset} dataset {model} Best Params,
{feature_count}{features}, and {classes}: \n")
            print("Classification Report: ")
            print(classification_report(y_test, clf_pred))

            # Best parameter set
            print(f'Best parameters found for {model}:\n', clf.best_params_)

            # Save model
            filename =
f'{dataset}_{model}_model_{feature_count}{features}_{classes}.sav'
            pickle.dump(clf, open(f'{directory}/{filename}', 'wb'))

        elif model == "Random_Forest":
            # Hyper-parameter space
            '''
            parameter_space = {
                'criterion': ['gini', 'entropy'],
                'n_estimators': [100, 200],
                "min_samples_split": np.linspace(0.1, 0.5, 3),
                "min_samples_leaf": np.linspace(0.1, 0.5, 3),
                "max_depth": [2, 10, 20],
                "max_features": ["log2", "sqrt", 'auto'],
            }
            '''
            clf = RandomForestClassifier()

            # Run Gridsearch
            # clf = GridSearchCV(clf, parameter_space, n_jobs=-1, cv=3)

            clf.fit(X_train, y_train)
            clf_pred = clf.predict(X_test)
            print(f"Test Results Using {dataset} dataset {model},
{feature_count}{features}, and {classes}: \n")
            print("Classification Report: ")
            print(classification_report(y_test, clf_pred))

            # Best parameter set
            # print(f'Best parameters found for {model}:\n',
clf.best_params_)
```

```
            # Save model
            filename = 
f'{dataset}_{model}_model_{feature_count}{features}_{classes}.sav'
            pickle.dump(clf, open(f'{directory}/{filename}', 'wb'))

        # Load Model
        filename = 
f'{dataset}_{model}_model_{feature_count}{features}_{classes}.sav'
        loaded_model = pickle.load(open(f'{directory}/{filename}', 'rb'))

        # Predict on validation set
        val_pred = loaded_model.predict(X_val)
        print(f"Validation Results Using {dataset} dataset, {model}, 
{feature_count}{features}, and {classes}: \n")
        print("Classification Report: ")
        print(classification_report(y_val, val_pred))
        cf_matrix = confusion_matrix(y_val, val_pred)
        print(cf_matrix)
        sns.heatmap(cf_matrix, annot=True, fmt="d")
        plt.title(f'{model} Confusion Matrix - {feature_count}{features}, 
{classes}')
        plt.show()

        # f1 scores for comparison table output
        f1_micro_class0 = f1_score(y_val, val_pred, average=None)[0]
        f1_micro_class1 = f1_score(y_val, val_pred, average=None)[1]
        f1_macro = f1_score(y_val, val_pred, average='macro')

        return f1_micro_class0, f1_micro_class1, f1_macro


    # Model already saved as external file
    else:
        # Load Model
        filename = 
f'{dataset}_{model}_model_{feature_count}{features}_{classes}.sav'
        loaded_model = pickle.load(open(f'{directory}/{filename}', 'rb'))

        # Predict on validation set
        val_pred = loaded_model.predict(X_val)
        print(f"Validation Results Using {dataset} dataset, {model}, 
{feature_count}{features}, and {classes}: \n")
        print("Classification Report: ")
        print(classification_report(y_val, val_pred))
        cf_matrix = confusion_matrix(y_val, val_pred)
        print(cf_matrix)
        sns.heatmap(cf_matrix, annot=True, fmt="d")
        plt.title(f'{model} - {feature_count} {features}, {classes}')
        plt.tight_layout()
        plt.show()

        if model == 'MLP' or model == 'Logistic_Regression':
            print(f'Best parameters found for {model}:\n', 
loaded_model.best_params_)

        # f1 scores for comparison table output
        f1_micro_class0 = f1_score(y_val, val_pred, average=None)[0]
        f1_micro_class1 = f1_score(y_val, val_pred, average=None)[1]
        f1_macro = f1_score(y_val, val_pred, average='macro')

        return f1_micro_class0, f1_micro_class1, f1_macro
```

Contextual_Ensemble_Model.py

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier, VotingClassifier,
GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, confusion_matrix, f1_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
import json
import os
import warnings
warnings.filterwarnings("ignore")

from project_root import get_project_root
root = get_project_root()

# parameters for file storage name
dataset = 'contextual'
model = 'Ensemble'
feature_count = 144
features = 'All_Features'
classes = 'all_classes'

if feature_count == 144:
    feature_count = ''
else:
    feature_count = feature_count


# data
df = pd.read_csv(root / '1.Data' / 'Contextual_data.csv')
df = df.drop(columns=['long', 'lat', 'Point'])
cols_to_move = ['Label']
df = df[cols_to_move + [col for col in df.columns if col not in
cols_to_move]]
# df = df[df['Label'].isin([0, 1])]

# Move Target to first column
target = 'Label'
first_col = df.pop(target)
df.insert(0, target, first_col)

# set features and target
X = df.values[:, 1:]
y = df.values[:, 0]
```

```
# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.25,
                                                  random_state=42)  # 0.25 x
0.8 = 0.2

# Feature Scaling
sc = StandardScaler()
sc.fit(X_train)
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)
X_val = sc.transform(X_val)



# models for ensembling
clf1 = MLPClassifier(hidden_layer_sizes = (100, 100, 100),
                     activation = 'tanh',
                     solver = 'adam',
                     alpha = 0.001,
                     learning_rate = 'invscaling')
clf2 = MLPClassifier(hidden_layer_sizes = (100, 100, 100),
                     activation = 'tanh',
                     solver = 'adam',
                     alpha = 0.001,
                     learning_rate = 'invscaling')
clf3 = MLPClassifier(hidden_layer_sizes = (100, 100, 100),
                     activation = 'tanh',
                     solver = 'adam',
                     alpha = 0.001,
                     learning_rate = 'invscaling')
clf4 = MLPClassifier(hidden_layer_sizes = (100, 100, 100),
                     activation = 'tanh',
                     solver = 'adam',
                     alpha = 0.001,
                     learning_rate = 'invscaling')
clf5 = MLPClassifier(hidden_layer_sizes = (100, 100, 100),
                     activation = 'tanh',
                     solver = 'adam',
                     alpha = 0.001,
                     learning_rate = 'invscaling')
clf6 = MLPClassifier(hidden_layer_sizes = (100, 100, 100),
                     activation = 'tanh',
                     solver = 'adam',
                     alpha = 0.001,
                     learning_rate = 'invscaling')
clf7 = MLPClassifier(hidden_layer_sizes = (100, 100, 100),
                     activation = 'tanh',
                     solver = 'adam',
                     alpha = 0.001,
                     learning_rate = 'invscaling')
clf8 = MLPClassifier(hidden_layer_sizes = (100, 100, 100),
                     activation = 'tanh',
                     solver = 'adam',
                     alpha = 0.001,
                     learning_rate = 'invscaling')
clf9 = MLPClassifier(hidden_layer_sizes = (100, 100, 100),
                     activation = 'tanh',
                     solver = 'adam',
                     alpha = 0.001,
                     learning_rate = 'invscaling')

# ensemble models
eclf1 = VotingClassifier(estimators=[('mlp1', clf1), ('mlp2', clf2), ('mlp3',
clf3), ('mlp4', clf4), ('mlp5', clf5), ('mlp6', clf6), ('mlp7', clf7),
('mlp8', clf8), ('mlp9', clf9)], voting='hard')

# Fit model
eclf1.fit(X_train, y_train)



# Predict on validation set
val_pred = eclf1.predict(X_val)
print(f"Validation Results Using Ensembled Contextual Models: \n")
print("Classification Report: ")
print(classification_report(y_val, val_pred))
cf_matrix = confusion_matrix(y_val, val_pred)
print(cf_matrix)
sns.heatmap(cf_matrix, annot=True, fmt="d")
plt.title(f'Contextual {model} Model - {feature_count}{features}, {classes}')
plt.show()

# f1 scores for comparison table output
f1_micro_class0 = f1_score(y_val, val_pred, average=None)[0]
f1_micro_class1 = f1_score(y_val, val_pred, average=None)[1]
f1_macro = f1_score(y_val, val_pred, average='macro')

# Save model
filename = f'{dataset}_{model}_model_{feature_count}{features}_{classes}.sav'
pickle.dump(eclf1, open(root / '1.Data' / filename, 'wb'))
```