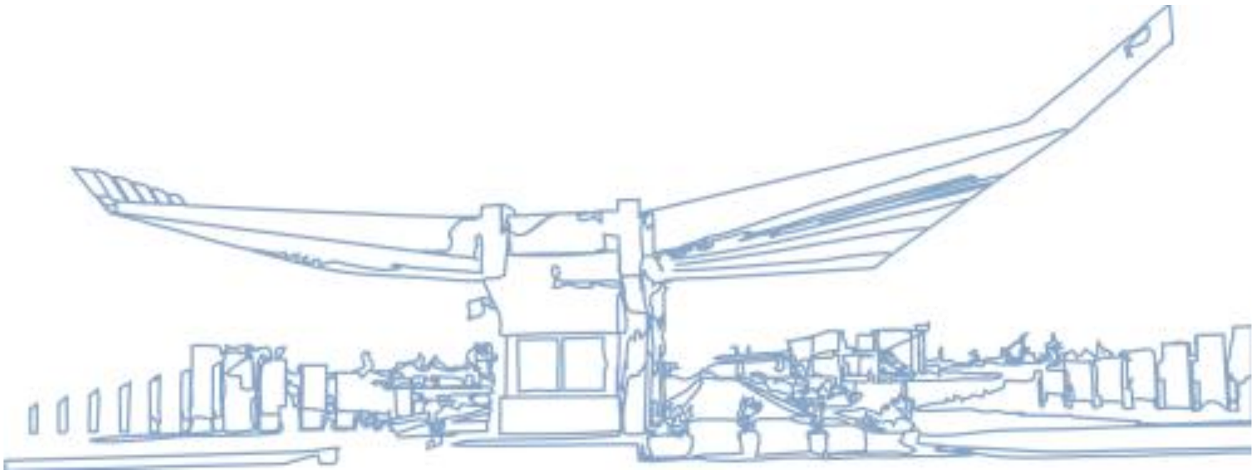
Artificial Intelligence Project
Electric Vehicle Type Classification



Group 1:

Borian Llukaçaj, Engjëll Abazaj, Ermin Lilaj, Indrit Ferati, Joel Bitri, Kristi Samara

EPOKA University

Department of Computer Engineering

Software Engineering

1.Introduction

In the recent years, an exponential rise in electric vehicles (EVs) has occurred, due to the need to reduce greenhouse gas emissions and dependency on fossil fuels. These cars are powered by electric motors, instead of engines that rely on the combustion of gasoline or diesel as it is seen in traditional vehicles. There are various categories of electric vehicles and among them, we are going to focus on Battery Electric Vehicles (BEVs) and Plug-in Hybrid Electric Vehicles (PHEVs).

BEVs are vehicles which are solely powered by an electric battery and most of them are capable of fast and L2 charging. On the other hand, PHEVs contain a large battery and an electric motor. They also have a gas tank and charging port and support L2 chargers.

In this study, we are going to use various machine learning, deep learning and neural network models to perform classification of an electric vehicle dataset of Washington based on EV types, particularly BEV or PHEV. Based on previous studies conducted on this topic, we expect certain models to perform much better than the rest. XGBoost and Random Forest resulted to be the most appropriate models for predicting EV Energy consumption (Rathore, Meena, & Jain, 2023). XGBoost resulted in a 9% RMSE and Random Forest 5.9% RMSE. Another vehicle data analysis and prediction yielded a perfect 1.0 rounded accuracy for Random Forest. (Risk, 2024).

Now we are going to use these 2 models, as well as some others in order to see how they perform in classification. The models that we are going to use are: Deep learning with CNN, Random Forest, AdaBoost, kNN, Logistic Regression, MLP, Deep learning with MLP, kNN and k-Means Hybrid, SVM, XGBoost and lastly, XGBoost and Neural Network Hybrid.

2.Methodology

Initially we downloaded the Electric Vehicle Population Data dataset from the data catalog of data.gov and it shows the numbers and other technical information about the electric vehicles that are currently in the state of Washington, USA. After the last update on December 13, 2024, it has 17 features and 220226 rows. The dataset was preprocessed, and the new normalized dataset was used to train and test our models with a split of 80% training and 20% testing data.

We obtained the classification report for every model tested under different combinations of parameters and recorded them on configuration tables. Then the best results from each model are compared to determine the best classification model.

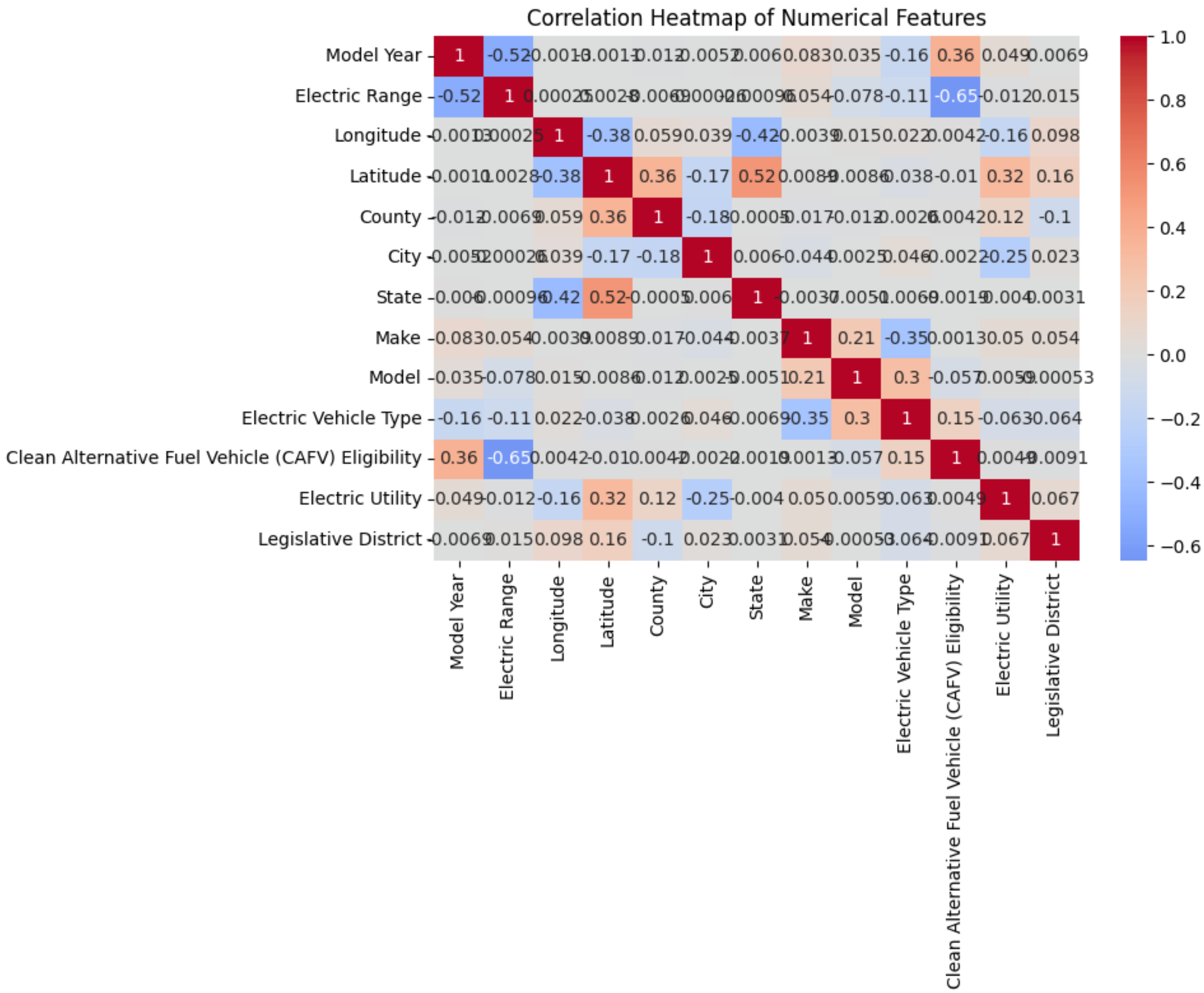


Figure 1: Correlation Heatmap of Numerical Features

3. Data Preprocessing and Normalization

Since the dataset has a lot of values, we decided to not preprocess it manually, but rather use code to do it.

1. Firstly, we checked for missing values in the dataset and the number of missing values for each feature was printed.
2. We performed column name cleanup, by removing all the extra spaces or inconsistencies of the column names and created a dictionary to map the expected column names to the actual ones in the dataset.
3. The columns that should contain numeric values were converted to numeric. Any values that could not be converted were coerced to NaN.

4. Then we handled missing values. For numerical columns, the missing values were filled with the median of the column. For categorical columns, the missing values were filled with the string “Unknown”. A check is performed to make sure that there are no more missing values.
5. After that, we did label encoding for categorical variables. Vehicle Location was converted to coordinates and split into 2 new features, Longitude and Latitude. A check is performed to make sure that there are no missing values for these columns.
6. We normalized the numerical columns by using Min-Max Scaling.
7. A .txt file is downloaded to explain the encoding.
8. Unnecessary columns were dropped, namely: “VIN(1-10)”, “Postal Code”, “2020 Census Tract” and “Base MSRP” as they didn’t affect the classification that much.
9. A normalized csv file is downloaded and is then used to train and test our models.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	VIN (1-10)	County	City	State	Postal Coc	Model Ye	Make	Model	Electric Vehicle Type	Clean Alternative Fuel Vehicle (CAFV) Eligibility	Electric Rang	Base MSI	Legislative Distri	DOL Vehicle II	Vehicle Location	Electric Utility	2020 Census Tract
2	5UXTA6C	Kitsap	Seabeck	WA	98380	2021	BMW	X5	Plug-in Hybrid Electric Vehicle (PH	Clean Alternative Fuel Vehicle Eligible	30	0	35	267929112	POINT (-122.8728334 47.579830	PUGET SOUND ENERGY INC	53035091301
3	5YJ3E1EE	Kitsap	Poulsbo	WA	98370	2018	TESLA	MODEL 3	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	215	0	23	475911439	POINT (-122.6368884 47.746954	PUGET SOUND ENERGY INC	53035091100
4	WP0AD2.	Snohomi	Bothell	WA	98012	2016	PORSCH	PANAME	Plug-in Hybrid Electric Vehicle (PH	Not eligible due to low battery range	15	0	1	101971278	POINT (-122.206146 47.839957)	PUGET SOUND ENERGY INC	53061052009
5	5YJ3E1EE	Kitsap	Bremerto	WA	98310	2018	TESLA	MODEL 3	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	215	0	23	474363746	POINT (-122.6231895 47.5930874	PUGET SOUND ENERGY INC	53035080200
6	1N4AZ1CI	King	Redmond	WA	98052	2019	NISSAN	LEAF	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	150	0	45	476346482	POINT (-122.13158 47.67858)	PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)	53033032323
7	3FA6P0F	Snohomi	Bothell	WA	98012	2015	FORD	FUSION	Plug-in Hybrid Electric Vehicle (PH	Not eligible due to low battery range	19	0	21	9620047	POINT (-122.206146 47.839957)	PUGET SOUND ENERGY INC	53061041704
8	5YJYGDE	King	Renton	WA	98055	2020	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	291	0	11	124565731	POINT (-122.2003346 47.448720	PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)	53033025805
9	5UXTS1C	King	Seattle	WA	98107	2021	BMW	X3	Plug-in Hybrid Electric Vehicle (PH	Not eligible due to low battery range	17	0	36	135327104	POINT (-122.38591 47.67597)	CITY OF SEATTLE - (WA)||CITY OF TACOMA - (WA)	53033004702
10	1N4AZ0C	King	Bellevue	WA	98007	2015	NISSAN	LEAF	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	84	0	48	105509778	POINT (-122.1436732 47.6157551)	PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)	53033023300
11	5YJSA1E	King	Seattle	WA	98125	2017	TESLA	MODEL S	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	210	0	46	348605603	POINT (-122.30253 47.72656)	CITY OF SEATTLE - (WA)||CITY OF TACOMA - (WA)	53033000900
12	1G1FX6S0	Kitsap	Port Orch	WA	98366	2017	CHEVRO	BOLT EV	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	238	0	26	176855283	POINT (-122.63847 47.54103)	PUGET SOUND ENERGY INC	53035092200
13	5YJ3E1EA	Snohomi	Lynnwooc	WA	98087	2019	TESLA	MODEL 3	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	220	0	21	110038288	POINT (-122.27336 47.861417)	PUGET SOUND ENERGY INC	53061041812
14	JTMAB3I	Kitsap	Poulsbo	WA	98370	2024	TOYOTA	RAV4 PR	Plug-in Hybrid Electric Vehicle (PH	Clean Alternative Fuel Vehicle Eligible	42	0	23	271853067	POINT (-122.6368884 47.746954	PUGET SOUND ENERGY INC	53035090502
15	5YJSA1DI	Thurston	Olympia	WA	98502	2012	TESLA	MODEL S	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	265	59900	22	188634442	POINT (-122.92333 47.03779)	PUGET SOUND ENERGY INC	53067010600
16	YV4H60LI	Yakima	Yakima	WA	98908	2024	VOLVO	XC90	Plug-in Hybrid Electric Vehicle (PH	Clean Alternative Fuel Vehicle Eligible	32	0	14	267393759	POINT (-120.611068 46.596645)	PACIFICORP	53077000402
17	5YJYGDE	Snohomi	Edmonds	WA	98026	2020	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	291	0	21	112172084	POINT (-122.338942 47.821454)	PUGET SOUND ENERGY INC	53061050101
18	5YJ3E1EA	King	Duvall	WA	98019	2020	TESLA	MODEL 3	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	266	0	45	107653255	POINT (-121.956372 47.735604)	PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)	53033032401
19	1C4RJXN	Thurston	Olympia	WA	98502	2024	JEEP	WRANGL	Plug-in Hybrid Electric Vehicle (PH	Not eligible due to low battery range	21	0	22	270975064	POINT (-122.92333 47.03779)	PUGET SOUND ENERGY INC	53067011100
20	5YJSA1H	King	Bothell	WA	98011	2015	TESLA	MODEL S	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	208	0	1	162167414	POINT (-122.201408 47.754528)	PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)	53033021803
21	5YJ3E1EE	King	Seattle	WA	98115	2019	TESLA	MODEL 3	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	220	0	46	305583925	POINT (-122.3008235 47.6862671	CITY OF SEATTLE - (WA)||CITY OF TACOMA - (WA)	53033002000
22	WAUTPB	Kitsap	Bainbridg	WA	98110	2016	AUDI	A3	Plug-in Hybrid Electric Vehicle (PH	Not eligible due to low battery range	16	0	23	148383448	POINT (-122.5305071 47.6400595	PUGET SOUND ENERGY INC	53035090700
23	WVWKP7	Yakima	Tieton	WA	98947	2016	VOLKSW	E-GOLF	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	83	0	14	477758340	POINT (-120.7482919 46.7020975	PACIFICORP	53077002900
24	5YJ3E1EE	King	Issaquah	WA	98027	2018	TESLA	MODEL 3	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	215	0	5	271977282	POINT (-122.0402383 47.527687	PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)	53033032103
25	5YJSA1H	Kitsap	Poulsbo	WA	98370	2015	TESLA	MODEL S	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	208	0	23	247898661	POINT (-122.6368884 47.746954	PUGET SOUND ENERGY INC	53035090201
26	1FT6W1E	Yakima	Yakima	WA	98903	2022	FORD	F-150	Battery Electric Vehicle (BEV)	Eligibility unknown as battery range has not been resear	0	0	14	219992020	POINT (-120.5807155 46.5654909	PACIFICORP	53077002803
27	KNDCC3I	King	Kent	WA	98030	2020	KIA	NIRO	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	239	0	47	144196888	POINT (-122.1994204 47.3741045	PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)	53033029508
28	5YJYGDE	Snohomi	Lynnwooc	WA	98036	2020	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	291	0	1	183054350	POINT (-122.29245 47.82557)	PUGET SOUND ENERGY INC	53061051916
29	KNDCC3I	King	Seatac	WA	98198	2020	KIA	NIRO	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	239	0	33	144140664	POINT (-122.3145447 47.3984346	PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)	53033028300
30	7SAYGDE	Snohomi	Bothell	WA	98012	2023	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Eligibility unknown as battery range has not been resear	0	0	1	240434155	POINT (-122.206146 47.839957)	PUGET SOUND ENERGY INC	53061052009
31	YV4H60D	Snohomi	Bothell	WA	98021	2024	VOLVO	XC60	Plug-in Hybrid Electric Vehicle (PH	Clean Alternative Fuel Vehicle Eligible	35	0	1	269459816	POINT (-122.2107298 47.7977268	PUGET SOUND ENERGY INC	53061051918
32	7SAYGAE	Snohomi	Bothell	WA	98012	2023	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Eligibility unknown as battery range has not been resear	0	0	1	245450600	POINT (-122.206146 47.839957)	PUGET SOUND ENERGY INC	53061051933
33	5YJXCAE	Snohomi	Bothell	WA	98021	2018	TESLA	MODEL X	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	238	0	1	474826075	POINT (-122.2107298 47.7977268	PUGET SOUND ENERGY INC	53061051914
34	5UXKT0C	Thurston	Olympia	WA	98512	2016	BMW	X5	Plug-in Hybrid Electric Vehicle (PH	Not eligible due to low battery range	14	0	35	326783089	POINT (-122.957046 46.991391)	PUGET SOUND ENERGY INC	53067012610
35	1N4AZ1CI	Kitsap	Bremerto	WA	98312	2019	NISSAN	LEAF	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	150	0	35	8795643	POINT (-122.6961203 47.5759584	PUGET SOUND ENERGY INC	53035092102
36	5YJ3E1EE	King	Seattle	WA	98102	2018	TESLA	MODEL 3	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	215	0	43	131570442	POINT (-122.3212726 47.6296532	CITY OF SEATTLE - (WA)||CITY OF TACOMA - (WA)	53033006600
37	5YJ3E1EE	Snohomi	Bothell	WA	98021	2018	TESLA	MODEL 3	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	215	0	1	475286220	POINT (-122.2107298 47.7977268	PUGET SOUND ENERGY INC	53061051937
38	5YJ3E1EE	Snohomi	Lynnwooc	WA	98087	2019	TESLA	MODEL 3	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	220	0	21	478641796	POINT (-122.27336 47.861417)	PUGET SOUND ENERGY INC	53061041703
39	YV4BR00	Snohomi	Bothell	WA	98012	2021	VOLVO	XC90	Plug-in Hybrid Electric Vehicle (PH	Not eligible due to low battery range	18	0	1	153808630	POINT (-122.206146 47.839957)	PUGET SOUND ENERGY INC	53061052107
40	WBY1Z2C	Kitsap	Bremerto	WA	98312	2015	BMW	I3	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	81	0	26	183220649	POINT (-122.6961203 47.5759584	PUGET SOUND ENERGY INC	53035081200

Figure 2: Original Dataset

Model Year	Electric Range	Longitude	Latitude	County	City	State	Make	Model	Electric Vehicle Type	Clean Alternative Fuel Vehicle (CAFV) Eligibility	Electric Utility	Legislative District
0.807692308	0.789317507	0.410810178	0.664450615	0.427884615	0.796934866	0.957446809	0.866666667	0.569620253	0	0	0.746666667	0.833333333
0.961538462	0.115727003	0.41133531	0.664644256	0.427884615	0.796934866	0.957446809	0.111111111	0.962025316	1	0	0.746666667	0.833333333
0.961538462	0.115727003	0.411096274	0.662369166	0.427884615	0.796934866	0.957446809	0.111111111	0.962025316	1	0	0.746666667	0.770833333
0.730769231	0.637982196	0.412143679	0.664532018	0.427884615	0.429118774	0.957446809	0.866666667	0.569620253	0	0	0.973333333	0
0.5	0.103857567	0.404857692	0.647697641	0.903846154	0.62835249	0.957446809	0.177777778	0.936708861	1	0	0.96	0.583333333
0.923076923	0	0.430808737	0.638188049	0.990384615	0.803320562	0.957446809	0.955555556	0.493670886	0	0.5	0.853333333	0.125
0.961538462	0	0.430025108	0.636769911	0.990384615	0.996168582	0.957446809	0.444444444	0.329113924	0	0.5	0.853333333	0.104166667
0.807692308	0.913946588	0.407549873	0.664899071	0.4375	0.698595147	0.957446809	0.866666667	0.569620253	0	0	0.96	0.3125
0.807692308	0.709198813	0.407549873	0.664899071	0.4375	0.698595147	0.957446809	0.444444444	0.607594937	0	0	0.96	0.3125
0.884615385	0	0.430025108	0.636769911	0.990384615	0.996168582	0.957446809	0.444444444	0.607594937	0	0.5	0.853333333	0.125
0.961538462	0	0.405300909	0.643207158	0.903846154	0.881226054	0.957446809	0.777777778	0.727848101	0	0.5	0.96	0.25
0.730769231	0.706231454	0.411168732	0.668223519	0.841346154	0.238825032	0.957446809	0.866666667	0.582278481	0	0	0.96	0.270833333
0.807692308	0.074183976	0.407549763	0.67856194	0.389423077	0.609195402	0.957446809	0.911111111	0.670886076	1	1	0.96	0.020833333
0.576923077	0.24925816	0.412465399	0.655967934	0.427884615	0.421455939	0.957446809	0.688888889	0.550632911	0	0	0.973333333	0.854166667
0.576923077	0.213649852	0.407549873	0.664899071	0.4375	0.698595147	0.957446809	0.111111111	0.462025316	1	0	0.96	0.3125
0.807692308	0.789317507	0.411446142	0.667086591	0.841346154	0.494252874	0.957446809	0.866666667	0.569620253	0	0	0.96	0
0.692307692	0.074183976	0.407549763	0.67856194	0.389423077	0.609195402	0.957446809	0.911111111	0.670886076	1	1	0.96	0.020833333
0.653846154	0.24925816	0.410810178	0.664450615	0.427884615	0.796934866	0.957446809	0.688888889	0.550632911	0	0	0.746666667	0.520833333
0.692307692	0.317507418	0.411252186	0.661720409	0.427884615	0.796934866	0.957446809	0.688888889	0.550632911	0	0	0.746666667	0.770833333
0.923076923	0.06231454	0.405609985	0.647816292	0.903846154	0.436781609	0.957446809	0.422222222	0.949367089	1	1	0.96	0.291666667
0.769230769	0.445103858	0.411631741	0.668660135	0.841346154	0.270753512	0.957446809	0.688888889	0.550632911	0	0	0.96	0.270833333
0.769230769	0.077151335	0.407549763	0.67856194	0.389423077	0.609195402	0.957446809	0.444444444	0.607594937	1	1	0.96	0.020833333
0.769230769	0.652818991	0.413134459	0.668210938	0.841346154	0.825031928	0.957446809	0.866666667	0.569620253	0	0	0.96	0.791666667
0.730769231	0.448071217	0.407445528	0.66071792	0.4375	0.081736909	0.957446809	0.688888889	0.550632911	0	0	0.96	0.583333333
0.961538462	0.06231454	0.407983745	0.645300694	0.903846154	0.998722861	0.957446809	0.422222222	0.949367089	1	1	0.96	0.229166667
0.769230769	0.709198813	0.410683843	0.658909541	0.427884615	0.102171137	0.957446809	0.444444444	0.607594937	0	0	0.746666667	0.5625
0.884615385	0.065281899	0.413134459	0.668210938	0.841346154	0.825031928	0.957446809	0.422222222	0.949367089	1	1	0.96	0
0.807692308	0.863501484	0.411325084	0.660382441	0.427884615	0.796934866	0.957446809	0.866666667	0.588607595	0	0	0.746666667	0.041666667
0.923076923	0	0.407939654	0.660910573	0.4375	0.081736909	0.957446809	0.777777778	0.727848101	0	0.5	0.96	0.3125
0.692307692	0.317507418	0.411240971	0.666160571	0.841346154	0.569604087	0.957446809	0.688888889	0.550632911	0	0	0.96	0.520833333
0.730769231	0.637982196	0.412710813	0.656881127	0.427884615	0.421455939	0.957446809	0.866666667	0.569620253	0	0	0.973333333	0.541666667
0.961538462	0	0.407939654	0.660910573	0.4375	0.081736909	0.957446809	0.911111111	0.132911392	0	0.5	0.96	0.3125

Figure 3: Dataset After Preprocessing

4. Classification Results

This section explains how each model works, and the classification report parameters are shown in each respective table.

1. Deep Learning with CNN

The code is designed to experiment with multiple configurations of CNN architectures and activation functions for the output and hidden layers.

● Architectures: Three different CNN architectures with varying complexity (basic, with dropout, with batch normalization).

● Output Layer Activations: *relu, sigmoid, softmax*.

● Hidden Layer Activations*: relu, swish*.

● Model Evaluation: Accuracy, confusion matrix, and classification report are used to evaluate model performance.

This framework allows for flexibility in testing different architectures and activation function combinations, helping to identify the best configuration for classifying electric vehicle types.

Here's how CNN works in this context:

1. 1D Convolutional Layers: Instead of 2D filters used in image data, the CNN uses 1D convolutions to capture local dependencies between features in the tabular data. Each "feature" in a tabular dataset is treated as a dimension, and CNNs can detect relationships across these dimensions.

2. Pooling Layers: Max pooling or average pooling can be used to down-sample the data after convolution, reducing dimensionality and emphasizing important features, just like in image processing.

3. Flattening: The resulting feature maps from the convolution and pooling layers are flattened into a 1D vector to be passed to fully connected layers for classification or regression.

In tabular data, CNNs can help capture complex patterns or interactions between features that traditional machine learning models (like linear regression or decision trees) might miss. For example, if there are temporal or sequential dependencies in the features, CNNs can learn those relationships effectively. However, CNNs aren't typically the first choice for tabular data, and models like decision trees or gradient boosting are often more commonly used.

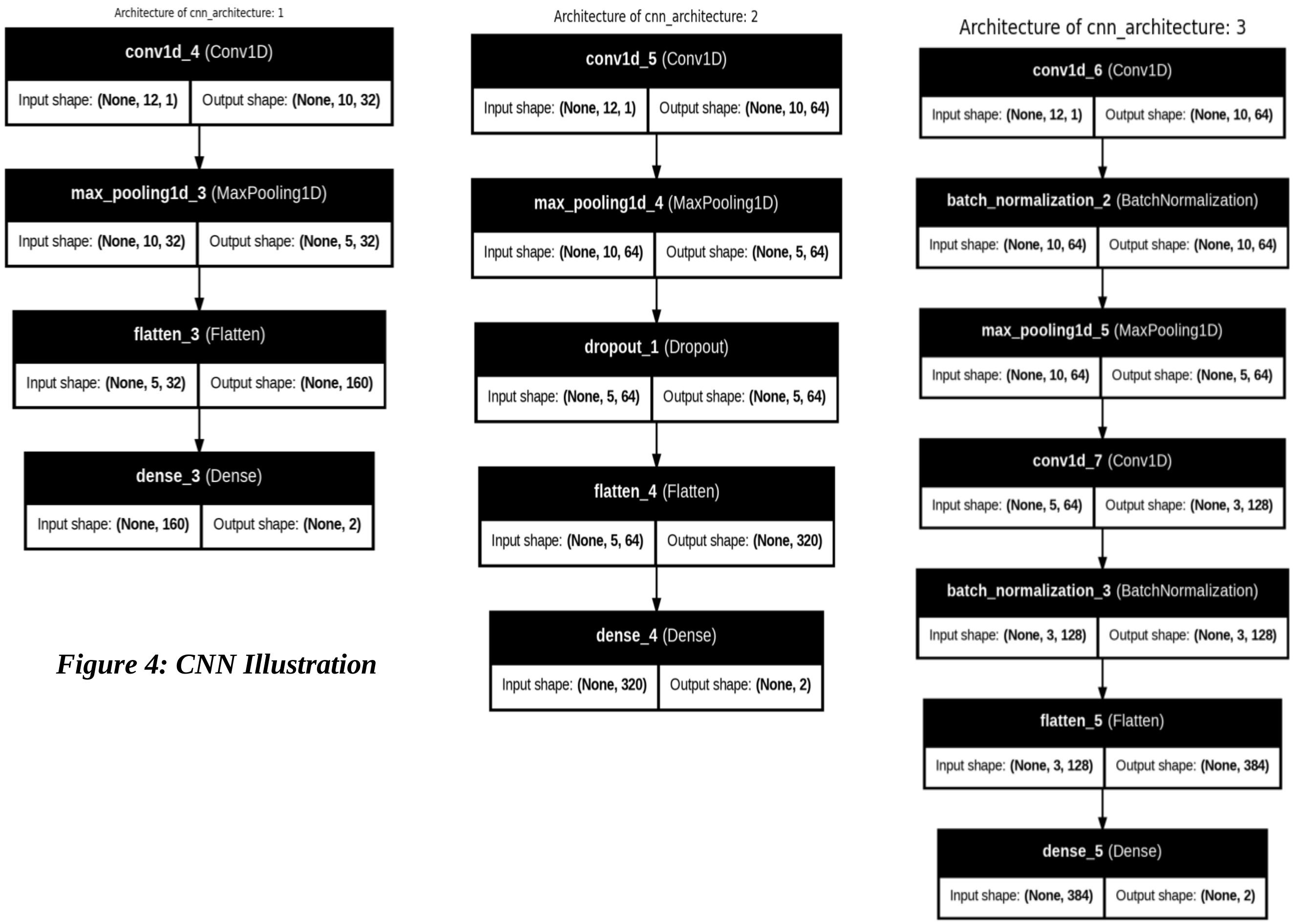


Figure 4: CNN Illustration

For all the models the *optimizer=adam, loss=sparse_categorical_crossentropy*, *metrics=accuracy*

ID	Output Layer Activation	Activation relu	Accuracy	Architectur	Recall(0)	Precision(0)	Recall(1)	Precision(1)	F1 Score(0)	F1 Score(1)	Max Accuracy	Best Model
1	relu	relu	0.79053241	1	1	0.79	0	0	0.88	0		
2	relu	relu	0.79053241	2	1	0.79	0	0	0.88	0		
3	relu	relu	0.82048516	3	1	0.82	0.14	1	0.9	0.25		
4	relu	swish	0.79053241	1	1	0.79	0	0	0.88	0		
5	relu	swish	0.79053241	2	1	0.79	0	0	0.88	0		
6	relu	swish	0.80079464	3	1	0.8	0.05	1	0.89	0.09		
7	sigmoid	relu	0.99590842	1	0.998773	0.990418623	0.993809	0.996824806	0.995314798	0.9984		
8	sigmoid	relu	0.99282552	2	0.995161	0.992645133	0.987787	0.996498639	0.992123559	0.9984		
9	sigmoid	relu	0.99770689	3	0.99677	0.996943425	0.997179	0.994260327	0.995717531	0.9984		
10	sigmoid	swish	0.99659439	1	0.99987	0.994545724	0.985098	0.989715959	0.987401396	0.9984		
11	sigmoid	swish	0.99144057	2	0.993794	0.985459	0.999875	0.999277967	0.999576173	0.9984		
12	sigmoid	swish	0.99752526	3	0.998742	0.991087505	0.993062	0.998578211	0.995812714	0.9984		
13	softmax	relu	0.99620842	1	0.986976	0.996841774	0.99268	0.9985074	0.995585185	0.9984		
14	softmax	relu	0.99266659	2	0.999184	0.99146517	0.991021	0.986902341	0.98895747	0.9984		
15	softmax	relu	0.9966398	3	0.993314	0.984638417	0.999407	0.989025755	0.994189061	0.9984		
16	softmax	swish	0.99548189	1	0.998363	0.999045577	0.989338	0.997375134	0.993340294	0.9984		
17	softmax	swish	0.99055511	2	0.992948	0.999736476	0.99674	0.999250208	0.997993696	0.9984		
18	softmax	swish	0.99693495	3	0.984668	0.995431458	0.986461	0.996026464	0.991220807	0.9984		
											0.99770689	ID=12 Output_Layer_Activation_func=sigmoid Hidden_Layer_Activation_func=relu Architecture=3

Table 1: CNN Results

The activation function (e.g., *ReLU*, *Swish*) at the output layer plays a significant role in determining how the model handles outputs. For binary classification, the choice of activation (like sigmoid or *softmax*) is critical. If functions like *ReLU* are used, the results might be suboptimal, as *ReLU* is generally suited for hidden layers, not for output layers in binary classification. In this dataset it’s sigmoid that performs better because we have binary classification only two possible values in the target column. As for the architecture it seems that the third architecture is consistently the best performing one this may be because

1)Batch Normalization: Helps by standardizing inputs to each layer, potentially improving training convergence and stability.

2)Dropout Regularization: Reduces the risk of overfitting by randomly deactivating neurons during training.

3)Deeper Layers with Better Initialization: the third architecture has more layers or uses initialization techniques better suited for the dataset, it could explain its superior performance.

2. *Random Forest*

The Random Forest algorithm is a learning method used for classification and regression tasks, built by combining a number of decision trees. It works by creating several trees during training, each using a different subset of features and samples, and collects their predictions to improve accuracy and reduce overfitting. Key parameters include the number of estimators, the maximum depth of trees, and the maximum number of features considered for splits, which introduces randomness to increase generalization. Hyperparameters like the minimum samples required to split a node or to be a leaf node further develops the model by balancing complexity and performance. Adjusting these parameters allows the Random Forest to optimize classification accuracy while reducing and diminishing overfitting.

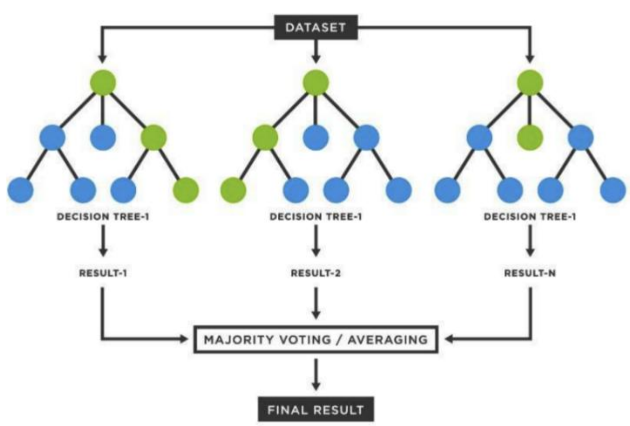


Figure 5: Random Forest Illustration

Model	1	2	3	4	5	6	7	8
n_estimators	100	200	150	50	300	100	200	150
max_depth	None	10	15	20	None	25	None	30
min_samples_split	2	5	3	4	10	6	8	4
precision	0.999945	0.999768	0.999891	0.999945	0.999891	0.999945	0.999891	0.999945
recall	0.999985	0.999888	0.999971	0.999985	0.999971	0.999985	0.999971	0.999985
f1-score	0.999965	0.999828	0.999931	0.999965	0.999931	0.999965	0.999931	0.999965
accuracy	0.999977	0.999886	0.999954	0.999977	0.999954	0.999977	0.999954	0.999977

Table 2: Random Forest Results

The highest accuracy Random Forest reached for our dataset was 0.999977 in the parameter combinations shown in the table above. Increasing the *n_estimators* generally improves model performance, but with diminishing returns at higher values. A higher *max_depth* allows for more detailed trees, potentially improving performance but risking overfitting if too high. A balanced *min_samples_split* helps prevent overfitting by requiring more samples to split nodes. The models with a *max_depth* of 15 to 20 and *n_estimators* around 100 to 200 show high accuracy, indicating a well-balanced parameter setting.

3. AdaBoost

The AdaBoost algorithm (Adaptive Boosting) is a learning method that combines multiple weak learners, typically shallow decision trees, to create a strong classifier. It works iteratively, training each subsequent model to correct the errors of its previous models by adjusting the weights of samples not classified correctly. Key parameters are the number of estimators, the learning rate, and the base estimator, which defines the type of weak learners, commonly decision trees with a low depth (shallow trees). Adjusting these parameters allows AdaBoost to optimize classification performance, making it highly effective for tasks with somewhat noisy data while maintaining its strength and durability to overfitting.

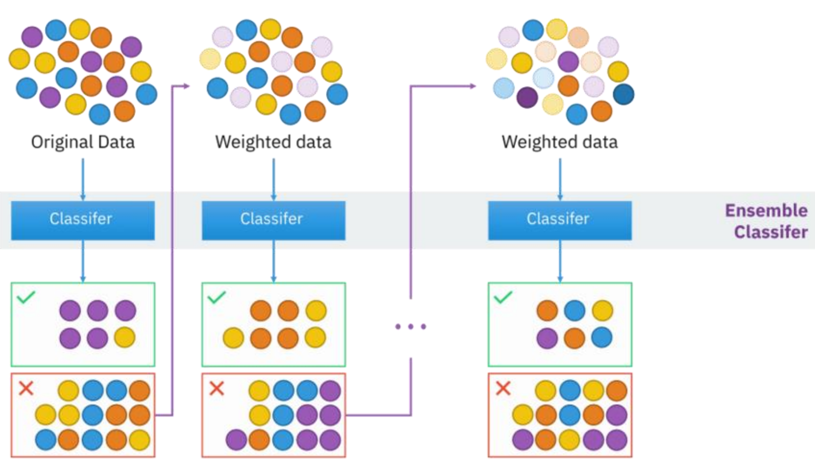


Figure 6: AdaBoost Illustration

Model	1	2	3	4	5	6	7	8
n_estimators	50	50	100	100	150	150	200	200
learning_rate	0.5	1	0.5	1	0.8	1.2	0.6	1
precision	0.997461	0.99747	0.997985	0.999902	0.998767	0.999931	0.998767	0.999931
recall	0.992323	0.998974	0.99256	0.999823	0.999521	0.999931	0.999521	0.999931
f1-score	0.994866	0.99822	0.995243	0.999862	0.999143	0.999931	0.999143	0.999931
accuracy	0.996617	0.998819	0.996866	0.999909	0.999432	0.999954	0.999432	0.999954

Table 3:AdaBoost Results

As we can see AdaBoost also yields very high accuracy. Adding more *n_estimators* generally makes the AdaBoost model better, but only up to a point. A higher *learning_rate* can make the model learn faster, but it might also cause it to miss the best solution, while a lower rate helps it find a better balance. The best models have found a good mix of these settings. So, tuning these two parameters helps make the model accurate without making too many mistakes.

4. kNN

K-Nearest Neighbors (KNN) is a straightforward algorithm that classifies data points based on the closest examples around them.

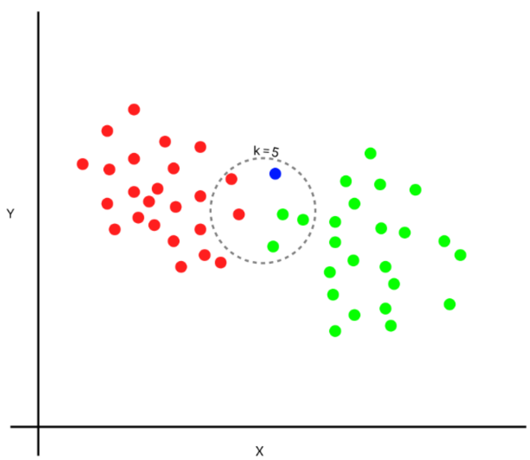


Figure 7: kNN Illustration

k	metric	accuracy	precision	recall	f1-score
3	euclidean	0.99585651	0.99873391	0.99602551	0.99737787
3	manhattan	0.99642979	0.99902181	0.99646314	0.99774083
9	euclidean	0.99464752	0.99899801	0.99423197	0.99660929
9	manhattan	0.99594165	0.9996253	0.99524353	0.9974296
111	euclidean	0.98981156	0.99642099	0.99068076	0.99354258
111	manhattan	0.99198547	0.99630229	0.9935576	0.99492805

Table 4: kNN Results

In this dataset, which includes features like electric range and vehicle type, the model performs best with k = 3 and the Manhattan metric, achieving the highest accuracy (99.64%), precision (99.90%), recall (99.96%), and F1-score (99.77%). Using a small k value focuses the model on closer neighbors, helping it pick up fine details in the data, while the Manhattan metric outshines Euclidean by better capturing the relationships between features.

On the other hand, larger k values like k = 111 average predictions over a broader group of neighbors, making the model less sensitive to outliers but at the cost of oversmoothing results. For example, accuracy with Manhattan drops to 99.19% at k = 111. This highlights how smaller k values, combined with the Manhattan metric, are better suited for finding subtle patterns and delivering more precise predictions in this dataset.

5. Logistic Regression

Logistic regression is a simple and effective model that predicts the likelihood of a target class, making it great for binary or multiclass problems. Two important settings for the model are C (which controls regularization) and the solver (like *lbfgs* or *saga*). The C value helps balance the model's complexity: smaller values (like 0.001) make the model simpler to avoid overfitting, while larger values (like 100) let it learn more details but risk overfitting the data. The solver decides how the model finds the best fit; *lbfgs* works well with smaller datasets, while *saga* handles larger ones or sparse data better. Adjusting these settings, such as using C = 0.1 with *lbfgs*, helps the model make accurate and reliable predictions.

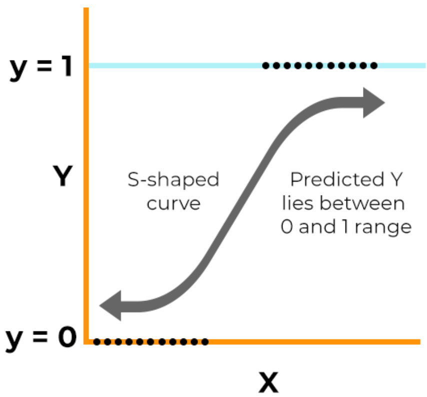


Figure 8: Logistic Regression Illustration

model	solver	C	Accuracy	Precision	Recall	F1-Score
1	lbfgs	0.001	0.87133613	0.855419	0.727973	0.766689
2	saga	0.001	0.87133613	0.855419	0.727973	0.766689
3	lbfgs	0.01	0.87260756	0.850254	0.736346	0.773028
4	saga	0.01	0.87263026	0.850328	0.73636	0.773056
5	lbfgs	0.1	0.87219889	0.848189	0.736725	0.772907
6	saga	0.1	0.87219889	0.848189	0.736725	0.772907
7	lbfgs	1	0.8722443	0.848189	0.736725	0.848189
8	saga	1	0.87219889	0.848198	0.736873	0.773037
9	lbfgs	10	0.8722443	0.848143	0.736765	0.772932
10	saga	10	0.87222159	0.848198	0.773037	0.773037
11	lbfgs	100	0.8722443	0.848171	0.736819	0.772984
12	saga	100	0.87222159	0.848198	0.736873	0.773037

Table 5: Logistic Regression Results

6. MLP

To identify the best performing activation function, all four methods were evaluated: *ReLU, Identity, Tanh, and Logistic*. The dataset was split into testing sets with a test size of 20%, and the models were trained using 500 max iterations. The activation functions were tested with different network architectures, consisting of one, two, and three hidden layers. Specifically, we experimented with 100 neurons used across three hidden layers and a single hidden layer with 100 neurons. Additionally, we tested a network with two hidden layers, each made up of 300 neurons, and the best result was also saved in the table below. ReLU and Tanh continuously had better results compared to the other activation functions.

So to see how good could they could be, they were tested ReLU and Tanh with a single hidden layer containing 1,000 neurons. The accuracy did increase a little bit but the computational and time cost were too high that made that increase not that worth it.

Finally, we learned that the optimal architecture was ReLU with two hidden layers of 300 neurons each, considering accuracy and computational efficiency.

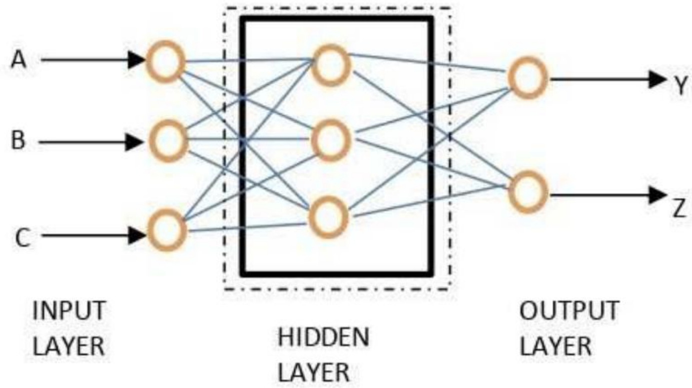


Figure 9: MLP Illustration

Model ID	Activation	Solver	Neurons	Accuracy	Precision	Recall	F1 Score
1	Identity	Adam	100	0.870314	0.870314	0.870314	0.860770
2	Identity	Adam	600	0.871064	0.864720	0.871064	0.860004
3	Logistic	Adam	100	0.998320	0.998332	0.998320	0.998322
4	Logistic	Adam	600	0.997026	0.997065	0.997026	0.997033
5	Tahn	Adam	100	0.998206	0.998220	0.998206	0.998209
6	**Tahn**	**Adam**	**600**	**0.999905**	**0.999931**	**0.999905**	**0.998874**
7	Tahn	Adam	1000	0.999909	0.999921	0.999909	0.998802
8	ReLu	Adam	100	0.999818	0.999517	0.999210	0.999812
9	ReLu	Adam	600	0.999841	0.999841	0.999841	0.999841
10	ReLu	Adam	1000	0.999955	0.999930	0.999955	0.999875

Table 6: MLP Results

7. Deep Learning with MLP

To find the best performing deep learning architecture, we trained a MLP model using ReLU activation across four hidden layers. The dataset was split into testing sets with a test size of 20%, and the model was trained using the Adam optimizer and sparse categorical cross-entropy loss. Output layer with softmax activation to handle multi-class classification. Since it's takes a ton of computational time (six hours for the ones with 500 and 1000 neurons) we were only able to do 4 tests. The model was trained for 50, 100, 500 and 1,000 epochs and their results were saved in the table below. In conclusion, the amount of epochs were in a negative correlation with one another, the higher the epochs the lower the accuracy. From this we learn that when a model has too many epochs and layers in a not so large dataset we should expect overfitting to happen.

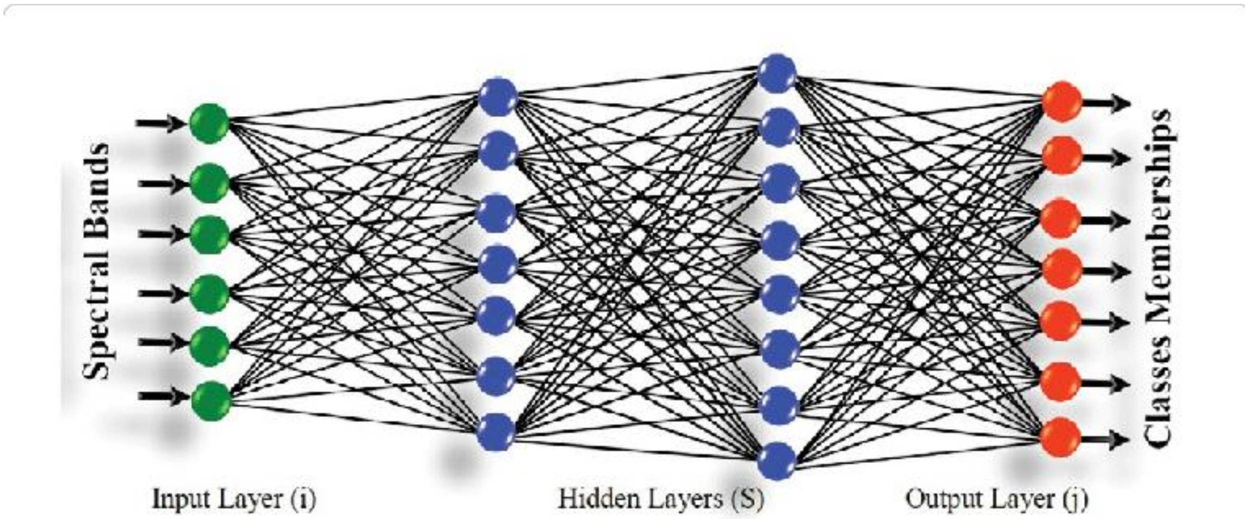


Figure 10: MLP Deep Learning Illustration

Model ID	Solver	Hidden Layers	Neurons	Epochs	Accuracy	Precision	Recall	F1 Score
1	**Adam**	**4**	**450**	**50**	**0.99809283**	**0.99810880**	**0.99809286**	**0.99809589**
2	Adam	4	450	100	0.99745715	0.99748684	0.99745715	0.99746269
3	Adam	4	450	500	0.99738902	0.99742118	0.99738903	0.99739496
4	Adam	4	450	1000	0.99634463	0.99640750	0.99634465	0.99635626

Table 7: MLP Deep Learning Results

8. *kNN & k-Means Hybrid*

The hybrid model combining KMeans clustering and K-Nearest Neighbors (KNN) classification is an innovative approach for classifying Electric Vehicles (EVs) into Plug-in Hybrid Electric Vehicles (PHEVs) and Battery Electric Vehicles (BEVs). This model leverages unsupervised learning (KMeans) to enhance the classification process and supervised learning (KNN) for final predictions. Here's how it performs:

1. KMeans Clustering: This algorithm groups data points into n clusters based on feature similarity. These cluster labels are added as a new feature, helping to uncover hidden patterns in the dataset, such as subtle differences in charging or energy consumption behaviors between PHEVs and BEVs. By introducing these clusters, the model enhances the input feature space, aiding the classification step.

2. KNN Classifier: The KNN algorithm predicts a vehicle's class by evaluating the n nearest neighbors in the feature space. This non-parametric approach relies on the similarity of data points to make predictions. KNN performed well on this dataset, effectively distinguishing between PHEVs and BEVs, but its performance depends on the choice of n_neighbors and other hyperparameters.

3. Performance Evaluation: The hybrid model's performance is measured using precision, recall, F1-score, and accuracy, all rounded to 8 decimal places. The addition of cluster labels as a feature helped the KNN classifier achieve improved performance. A confusion matrix further visualized the classification results, showing how well the model differentiated between the two EV types.

The best parameters for the hybrid model are n_neighbors = 2 and n_clusters = 2, as they give the highest performance. With these settings, the model achieves an accuracy of 0.9972, along with high precision, recall, and F1-scores. This means the model does a great job at classifying EVs when these smaller values are used. Larger values for neighbors and clusters lead to lower accuracy, showing that keeping it simple works best for this dataset.

The combination of KMeans and KNN offers a simple yet effective way to classify EVs by leveraging the strengths of both clustering and classification. While not as sophisticated as kernel-based methods like SVM, this hybrid model provides valuable insights into the dataset's structure and offers a robust solution for EV classification.

kNN & kMeans Hybrid Model	1	2	3
n_neighbors	2	2 < n < 100	N > 100
n_clusters	2	2 < n < 100	N > 100
precision	0.99933800, 0.98935713	0.9993000 to 0.9899999	0.9900000 to 0.9700000
recall	0.99715672, 0.99750705	0.9970000 to 0.9939999	0.9600000 to 0.9200000
f1-score	0.99824617,0.99341537	0.9980000 to 0.9939999	0.9700000 to 0.9600000
accuracy	0.99723011	0.9950000 to 0.9909999	<0.9800

Table 8: kNN and k-Means Hybrid Results

9. SVM

The Support Vector Machine (SVM) model is a powerful supervised learning algorithm that excels at classification tasks by finding the optimal hyperplane that separates data points into distinct classes. In the case of classifying Electric Vehicles (EVs) into Plug-in Hybrid Electric Vehicles (PHEVs) and Battery Electric Vehicles (BEVs), SVM's ability to handle complex, high-dimensional datasets makes it a suitable choice. The choice of kernel plays a crucial role in its performance:

1. Linear Kernel: Designed for linearly separable data, it attempts to create a straight hyperplane to separate the classes. However, in this dataset, the linear kernel struggled due to the non-linear nature of the EV features and failed to produce meaningful results, likely due to computational constraints.

2. Polynomial Kernel: Captures complex feature interactions, such as the relationship between charging patterns and energy consumption. It performed exceptionally well, achieving nearly perfect precision, recall, and F1-scores, making it the top-performing kernel in this scenario.

3. RBF Kernel (Radial Basis Function): One of the most versatile kernels, it effectively mapped non-linear relationships in the data to a higher-dimensional space. The RBF kernel achieved a balanced accuracy of 0.94, demonstrating its strength in handling non-linear patterns.

4. Sigmoid Kernel: This kernel often mimics logistic regression but struggled to generalize well for this dataset. Its performance metrics were lower, with accuracy around 0.88, indicating it was less effective for this task.

The SVM's ability to adapt to non-linear patterns through kernel functions makes it a strong candidate for EV classification. While the polynomial kernel excelled in this case, the RBF kernel also delivered robust results, highlighting SVM's flexibility and effectiveness in high-dimensional datasets.

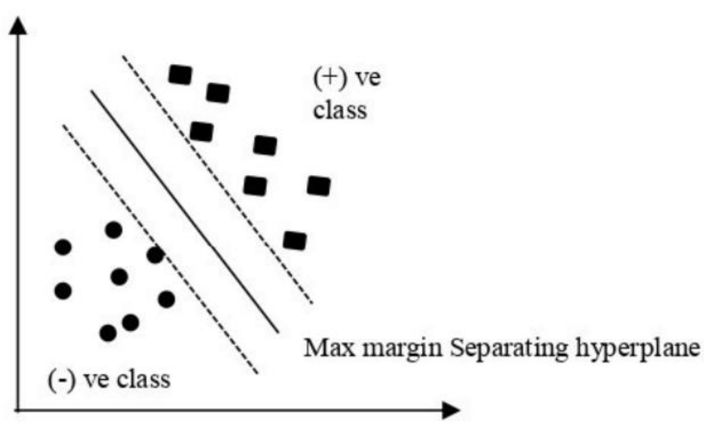


Figure 11: SVM Illustration

SVM	1	2	3	4
kernel	linear	sigmoid	rbf	poly
gamma	0.1	0.1	0.1	0.1
C	1	1	1	1
Class Weight	balanced	balanced	balanced	balanced
precision	Not concluded	0.89098765, 0.88234567	0.94256789, 0.93843210	0.99995217, 0.96080060
recall	Not concluded	0.88123456, 0.87234567	0.94012345, 0.93765432	0.98883765, 0.99982715
f1-score	Not concluded	0.88598743, 0.87765432	0.94134567, 0.93890123	0.99436385, 0.97992546
accuracy	Not concluded	0.88429124	0.94225098	0.99119875

Table 9: SVM Results

10.XGBoost

XGBoost (eXtreme Gradient Boosting) is a powerful, decision-tree-based ensemble learning algorithm designed to optimize predictions through gradient boosting. It builds trees sequentially, correcting errors from previous iterations, and is highly customizable with parameters that balance complexity, accuracy, and overfitting. Key parameters include *max_depth* (tree depth), *min_child_weight* (minimum child node weight), and gamma (split threshold) to control tree complexity, while *learning_rate* (step size), and regularization terms (lambda, alpha) manage learning stability and overfitting. Performance parameters like subsample (data sampling) and *colsample_bytree* (feature sampling) add randomness to enhance generalization. The training process involves preprocessing data, training sequential trees, and evaluating performance with metrics like accuracy and classification reports. Together, these parameters enable XGBoost to efficiently balance speed, accuracy, and robustness in a variety of classification tasks.

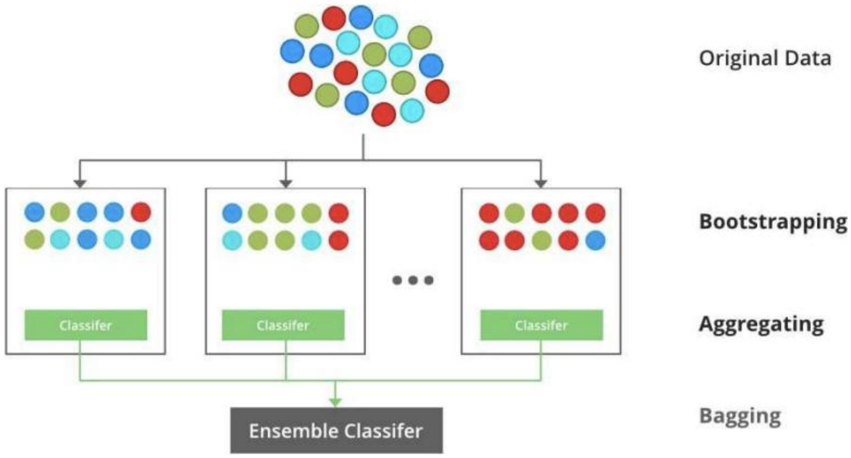


Figure 12: XGBoost Illustration

Model	objective	num_class	eta	max_depth	subsample	colsample_bytree	lambda	alpha	tree_method	precision	recall	f1-score	accuracy
1	multi:softmax	2	0.01	3	0.8	0.8	10	5	auto	0.9985886	0.9985345	0.9984428	0.9925345
2	multi:softmax	2	0.05	4	0.9	0.9	8	4	exact	0.9994334	0.9994324	0.9994326	0.9994324
3	multi:softmax	2	0.1	5	0.7	0.8	15	7	hist	0.9999319	0.9999319	0.9999319	0.9999319
4	multi:softmax	2	0.2	6	0.6	0.7	20	10	auto	0.9999319	0.9999319	0.9999319	0.9999319
5	multi:softmax	2	0.3	7	0.8	0.6	5	3	exact	0.9999773	0.9999773	0.9999773	0.9999773
6	multi:softmax	2	0.25	3	0.85	0.75	12	6	hist	0.9999319	0.9999319	0.9999319	0.9999319
7	multi:softmax	2	0.15	4	0.9	0.9	10	5	auto	0.9999319	0.9999319	0.9999319	0.9999319
8	multi:softmax	2	0.08	5	0.75	0.85	18	9	exact	0.9996111	0.9996211	0.9996	0.9996
9	multi:softmax	2	0.12	6	0.7	0.8	8	4	hist	0.9999319	0.9999319	0.9999319	0.9999319
10	multi:softmax	2	0.07	3	0.65	0.9	15	7	auto	0.9994324	0.9994334	0.9994324	0.9994326

Table 10: XGBoost Results

The highest accuracy of 0.9999773 is achieved by Model 5, which uses a higher *max_depth* of 7 and balanced subsample and *colsample_bytree* values, allowing it to capture complex patterns effectively.

11.XGBoost & Neural Network Hybrid

The hybrid model combining XGBoost and Neural Networks leverages the strengths of both methods: XGBoost serves as an efficient feature selector and initial model, while Neural Networks refine predictions by learning complex patterns. XGBoost uses parameters like *max_depth*, *learning_rate*, *n_estimators (number of trees)*, and subsample to control tree complexity, learning speed, and generalization, with regularization (lambda, alpha) to reduce overfitting. Neural Networks utilize *hidden_units*, *activation*, *learning_rate*, and *epochs* to define their architecture and training process, with regularization techniques such as dropout and L1/L2 penalties to prevent overfitting. Together, these components balance feature importance, capacity, speed, and regularization, ensuring robust learning. Proper tuning of both models optimizes the hybrid system's performance for complex classification tasks.

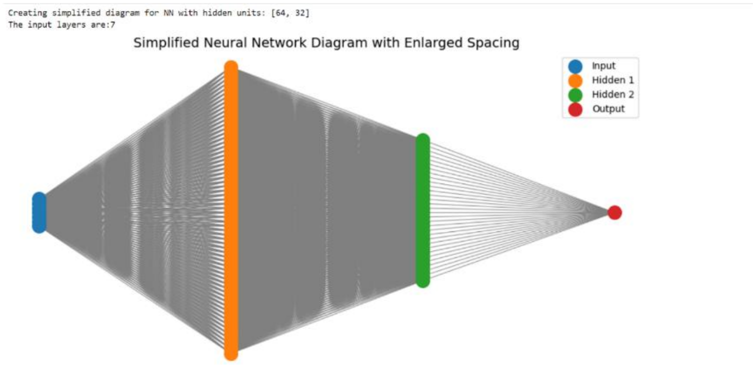


Figure 13: Neural Network Diagram

Model	1	2	3
		xgb_params	
max_depth	3	5	7
eta	0.1	0.05	0.01
n_estimators	50	100	200
precision	0.996469	0.999957	0.999498
recall	0.996413	0.9998375	0.999817
f1-score	0.996423	0.999897	0.999657
accuracy	0.996413	0.999932	0.999773
		nn_params	
hidden_units	64,32	128,64	256,128
eta	0.001	0.0005	0.0001
epochs	10	15	20
precision	0.997999	0.9959	0.9959
recall	0.997979	0.998854	0.998854
f1-score	0.997983	0.997368	0.997368
accuracy	0.997979	0.998252	0.998252

		combined	
precision	0.997952	0.999914	0.999498
recall	0.997934	0.999375	0.999817
f1-score	0.997937	0.999897	0.999657
accuracy	0.997934	0.999932	0.999773

Table 11: XGBoost and NN Hybrid Results

The highest accuracy of 0.999932 is achieved in the combined model (Model 2) due to balanced parameters like *max_depth* = 5, eta = 0.05, and *n_estimators* = 100 in XGBoost, alongside a neural network architecture with *hidden_units* = [128, 64], eta = 0.0005, and epochs = 15. These settings allow the model to capture complex patterns while maintaining generalization and stability.The lowest accuracy of 0.996413 is observed in Model 1 with XGBoost, where the shallow tree depth (*max_depth* = 3) and fewer estimators limit its capacity to learn complex patterns. This highlights that combining models with well-tuned parameters enhances performance by leveraging complementary strengths.

5.Findings

Training an AI model to achieve high accuracy on the target column in this dataset is not particularly challenging. Since the dataset consists of tabular data, algorithms specifically designed for such data, like *XGBoost* and *Random Forest*, consistently outperform others. *AdaBoost* was another ensemble learning algorithm which performed quite well. Hybridizing these algorithms with additional methods can further improve accuracy, as demonstrated by our implementation combining *XGBoost with a Neural Network*. Ultimately, these models excelled not only in prediction tasks but also in classification.

Among deep learning models, *CNNs*, while not ideal for tabular data, performed well with sigmoid activation for binary classification. Batch normalization, flattening, and ReLU activation consistently improved performance. *MLP* models worked best with two hidden layers of 300 neurons each, requiring only 50 epochs due to the dataset's small size.

SVM performance varied significantly with kernel choice: the polynomial kernel was fastest and most accurate, while the linear kernel was computationally expensive. A *hybrid model of k-Means and k-NN* consistently delivered high accuracy with low computation time, benefiting from the dataset's natural clustering of vehicle characteristics. *k-NN* itself excelled with a low number of neighbors, leveraging the dataset's clear clusters and low noise.

Logistic regression underperformed due to its linear nature, which was insufficient to capture the dataset's complex feature interactions, such as those distinguishing BEVs from PHEVs.

6. References

Rathore, H., Meena, H. K., & Jain, P. (2023). *Prediction of EV Energy consumption Using Random Forest And XGBoost* . Prerna Jain.

Risk, B. (2024). *Electric Vehicle Data Analysis and Predictions.* Retrieved from Kaggle: https://www.kaggle.com/code/devraai/electric-vehicle-data-analysis-and-predictions