Machine Learning

Final Project

FALL 2021

Annika Timermanis
Axel Solano
AmirHossein Hadinezhad

December 7th 2021

Prof. Andrew Delong

1. Introduction

We tried to perceive this project as an experiment in which we were trying to determine which classifier or which regressor was most suitable for a variety of datasets. We trained 8 models for each dataset, and observed their performance and errors. For the models that showed poor scores, we re-ran the training using more specific distribution domains for the search. We also compared the performance of a Convolutional Neural network and a Decision Tree on the same dataset, and tried to further analyze this with the activation maximization and novelty component. Our novelty component was inspecting our CNN filters from Part 3. We used a similar approach to what was done in Lab 9, but this time we dealt with 3 channels for RGB, and had many more layers to analyze.

2. Methodology & Experiments

Data Preprocessing

We imported each dataset using various import tools to load and store our data. We displayed each dataframe, and made changes where necessary. In the cases of NaN values, we used tools such as LabelEncoder[1]. In the case of unknown values, we replaced all '?' values to NaN, and then used the SimpleImputer, with strategy='mean'. We applied the SimpleImputer to both our training sets and testing sets. Originally we wanted to disregard those values for the training, however after we attempted to do so, our training size became far too small.

Training and Splitting Data

We used the train_test_split method from sklearn.model_selection to split up our data. We had originally attempted to split up our training and testing without shuffling the features which gave really strong predictions, but after revising techniques used in lab 7, we learned that even though our predictions might be weaker, it is more important to shuffle the features because it allows us to pick samples from all over, and therefore it is less likely that we are overfitting[2].

Hyperparameter Search

The prediction accuracy for the majority of models depended on specific hyperparameters. Hyperparameter values were not learned, rather had to be selected manually and tuned. For the tuning of hyperparameters we chose to use RandomizedGridSearch as opposed to UniformGridSearch, as suggested in the lecture on HyperParameter Search. For each model, we focused on tuning only 2 or 3 parameters.

[1] https://stackoverflow.com/questions/24458645/label-encoding-across-multiple-columns-in-scikit-learn
[2] https://stackoverflow.com/questions/57743639/why-do-i-get-different-results-when-i-do-a-manual-split-of-test-and-train-data-a

Testing & Success metrics

For testing, after each model was trained, we compared it's predictions on our X_test, against the true values y_test for that dataset. At the end of each dataset, the model that performed the best stood as a good indicator that it was a good model for that specific dataset. For classification, we used weighted f1 score, recall and recall for targeted class , and for regression we used RMSE and MSE. MSE is a good metric, as it gives us a good representation of how far our prediction is from our target. F1 score is a good metric because it takes the average of precision and recall which take into account imbalances in the data. We compared all evaluations on both training and test sets.

2.1 Classification

For each dataset listed below, we trained and tuned a Linear Regression classifier, a SVM classifier, a Decision Tree classifier, a Random Forest classifier, an AdaBoost classifier, a KNN classifier, a Gaussian Naive Bayes classifier, and a Neural Network classifier. In order to determine the best model for each of the datasets listed below, we focused on the f1 scores. Our data can be found on page 9.

Dataset	Best Model
1. Diabetic Retinopathy	Neural Network
2. Default of Credit Card Clients	Gaussian Naive Bayes
3. Breast Cancer Wisconsin	AdaBoost Classifier
4. Statlog (German credit data)	LogisticRegression
5. Adult	AdaBoost Classifier
6. Yeast	LogisticRegression
7. Thoracic Surgery Data	NAN
8. Seismic-Bumps	NAN

2.2 Regression

For each dataset listed below, we trained and tuned a Linear Regression model, a SVM regressor, a Decision Tree regressor, a Random Forest regressor, an AdaBoost regressor, a

KNN regressor, a Gaussian Naive Bayes regressor, and a Neural Network regressor. In order to determine the best model for each of the datasets listed below, we focused on the RMSE (root mean squared error) value, as well as the MSE (mean squared error) and the r2 scores. Our data can be found on page 11.

Dataset	Best Model
1. Wine Quality	---
a. Red Wine Quality	Random Forest Regressor
b. White Wine Quality	Random Forest Regressor
2. Communities and Crime	Linear Regression
3. QSAR aquatic toxicity	Support Vector Regression
4. Facebook metrics	Linear Regression
5. Bike Sharing (missing Gaussian Process Regressor and Neural Network)	*Random Forest Regression
6. Student Performance	Support Vector Regression
7. Concrete Compressive Strength	Random Forest Regressor
8. SGEMM GPU kernel performance	Inconclusive results, did not terminate.

* Best model out of the models that terminated.

2.3 Classifier interpretability - CIFAR-10

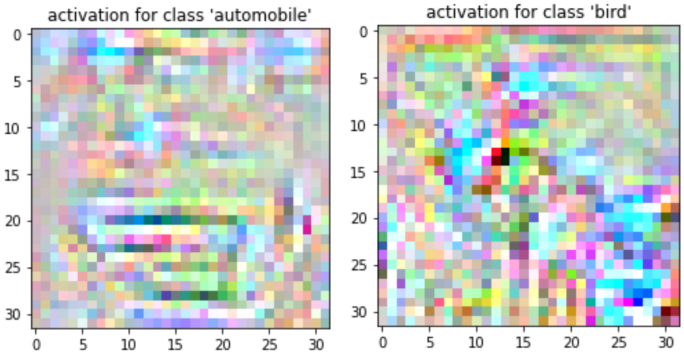
CNN

We used torch.tensor, rescaling our values from [0,255] to [-1,1][3], to create train and test sets. For the model itself, we used the same approach as seen in Lab 9, defining the parameters for our CNN. We first attempted using 3 conv2d layers, each with ReLU activations and 2 maxPool layers, and we kept the size of out_channels constant. Our model was trained relatively quickly, however its performance was only around 58%. We decided we needed additional hidden layers, as well as a larger number of out_channels, to allow the layers to potentially learn more useful features about the input[4].

[3] https://towardsai.net/p/data-science/how-when-and-why-should-you-normalize-standardize-rescale-your-data-3f083def38ff
[4] https://stackoverflow.com/questions/56652204/pytorch-convolution-in-channels-and-out-channels-meaning

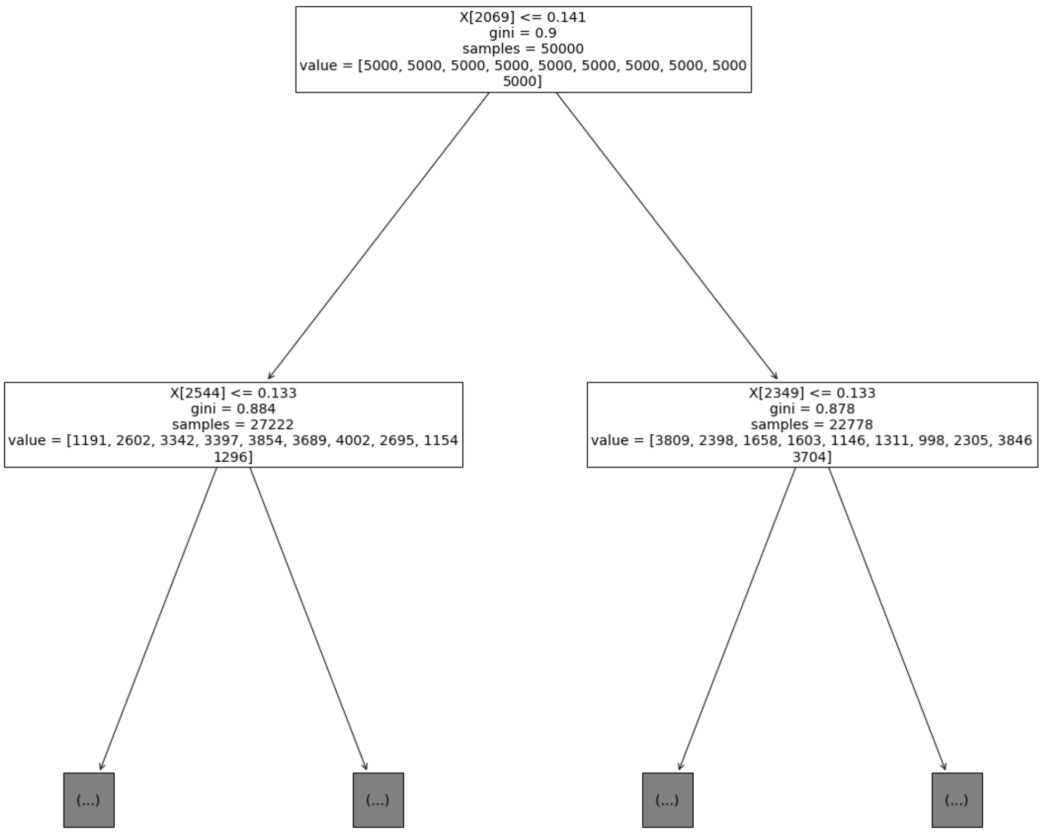
Activation Maximization

The activation-maximization was used to observe what input image would most strongly activate a particular output class prediction. It allowed us to get a better look at how a CNN can build features solely based on an input of pixels.



Decision Tree

Every image was reshaped into a vector of features which are the pixels of the image. The hyperparameters max depth and min sample split, were tuned using randomized search. We decided to tune those hyperparameters, and not the other hyperparameters such as min sample leaf because they similarly reduce overfitting. The results of the Decision Tree were not good. The accuracy score was 0.3092 on the test data. The decision tree has to choose one pixel feature that is most useful in differentiating the images into the ten classes.



The decision tree has a clear visual representation. If we follow the path/sequence of the nodes in the tree we would know which features were the most important and in which sequence. Therefore, it is somewhat interpretable. The model chose certain features at each node because they had the lowest impurity but the reason behind it is not obvious.

Interpretability of CNN vs Decision Tree

We believe that the CNN was more interpretable than the Decision Tree. In the activation maximization figure, we can almost see what objects led the CNN to classify an image as a particular class. In contrast, we cannot really tell what the final classification will be by looking at the features used in the nodes

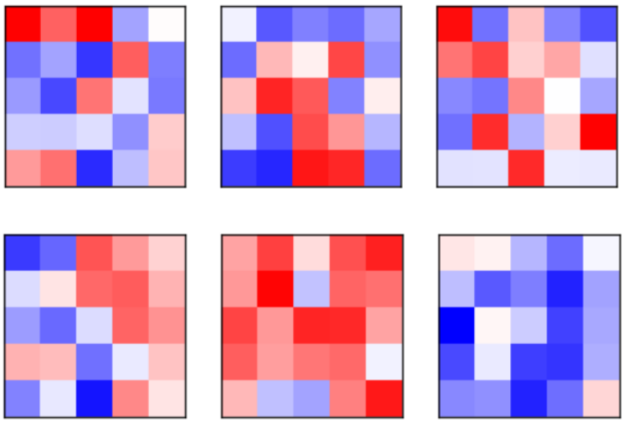
3. Novelty Component

Summary

For our novelty component, we chose to inspect the filters of our CNN. We wanted to specifically focus on comparing and contrasting the filter weights from our first layer, and our final layer. The early layers in our CNN tried to focus on broad features, while the later layers tried to detect specific features.

Observations

The first row of 3 filters are from our first input layer of the CNN, the second row of 3 filters is from our last layer of the CNN. The filter results we obtained were not as interpretable as we expected. We believe that it is hard to capture distinct patterns on only 5 by 5 pixel filters. The values in the filters from the first conv2d layer were more random, probably because they learned less complex features than the last conv2d layer which has less variation in the values.



4. Overall Observations & Reflection

The reason why we were not capable of having all 8 models trained for the GPU Kernel and Bike sharing datasets, as well as the improvements stated below, was due to losing our fourth team member late in the semester.

Improvements

- Tested all models with *different kinds of scaling* (e.g. distribution normalization, min-max normalization) and compared the results to see the impact that different scaling methods could have. We specifically use all different scaling on Breast Cancer Dataset and observed how they distribute our data with different plots , ideally we wanted to train different models on each to make sure we produce the best model in the end. Normalizing the data was used for the CNN in Part 3 but such scaling could have been used in the more on other datasets in order to speed up the training and be able to further tune the models. In addition, the models could have learned in a way that generalizes better because they give more equal weight to each input feature.
- We had implemented a *dummy estimator* in some files for baseline reference, but due to a lack of time, we were not able to re-run with the dummy estimator present. A dummy estimator is used to give a sample prediction, it does not give strong insight, however is better than a random guess[5]. Both a dummy regressor and dummy classifier could have been implemented for classification and regression datasets. The dummy classifier could have further helped anticipate unbalanced datasets[6].
- For the CNN for Part 3, we realize we should have started *running this earlier*. We were only able to achieve 71% accuracy. We had it training on a high number of epochs, with a large number of output_channels. As we reached the last epochs, the training accuracy started to lower, and we were not improving. We would have *trained on a smaller number of epochs and output_channels* to begin with, and with more time we could have had more chances of re-running with slight modifications and potentially achieve 75% accuracy.
- For the CNN for Part 3, we also would have wanted to try using *data augmentation techniques* with torchvision.transforms[7], as the model was taking a long time to train, we didn't have time to re-run on different sets of training input. We believe the data augmentation could have helped because, rather than try to make the model bigger in order to increase performance, and increase the number of epochs (which eventually makes the accuracy worse due to overfitting the training set), we could have tried increasing the amount of data we were training on. It is difficult to conclude on which augmentation techniques would have been best, but we would have wanted to try using different techniques such as CenterCrop, RandomAffine or Grayscale, to name a few.

[5] https://towardsai.net/p/data-science/dealing-with-class-imbalance%E2%80%8A-%E2%80%8A-dummy-classifiers
[6] https://www.geeksforgeeks.org/ml-dummy-classifiers-using-sklearn/
[7] https://pytorch.org/vision/master/transforms.html

5. Conclusions

We observed that for most datasets that involve regression used in this project, SVR is very reliable in terms of performance. However, it takes a long time to train and search for the right hyperparameters. Therefore, we think that linear regression models and random forest regressors have a good tradeoff between performance and training speed and should be used as a first option and SVR as a second option. For classification problems, Decision Trees showed good performance on various datasets. Given its fast training speed, this algorithm should be used before trying more complex models like SVM and Neural Networks that can perform very well but are difficult to train.

Most datasets if not all datasets should be normalized to improve performance and training speed.

Our novelty component involving inspecting the CNN filters did not improve the interpretability of the model. The filters did not show defined structures that could explain how the model learned the image features.

Results

For more analysis please refer back to each file as we provided more information for each model on the file. Below i the overall performance of them

Classification Results

acc = accuracy, w.f1 = weighted f1, w.rec = weighted recall, clas.rec = class recall, tr.acc = training accuracy, tr.w.f1 = training weighted f1, tr.w.rec = training weighted recall, tr.clas.rec = training class recall

1. Diabetic Retinopathy

model	acc	w.f1	w.rec	clas.rec	tr.acc	tr.w.f1	tr.w.rec	tr.clas.rec
gnb	0.606061	0.607531	0.606061	0.607692	0.646739	0.646253	0.646739	0.681913
logistic_r	0.714286	0.713375	0.714286	0.615385	0.744565	0.743216	0.744565	0.661123
nn	0.735931	0.735071	0.735931	0.784615	0.925	0.924979	0.925	0.933472
svc	0.731602	0.732206	0.731602	0.669231	0.81087	0.809568	0.81087	0.715177
forest	0.65368	0.654968	0.65368	0.623077	1	1	1	1
adaboost	0.705628	0.706478	0.705628	0.653846	0.848913	0.848905	0.848913	0.810811
knn	0.632035	0.629991	0.632035	0.530769	1	1	1	1
tree	0.619048	0.606847	0.619048	0.438462	0.658696	0.650242	0.658696	0.50104

GNB	Logistic	NN	SVC	FOREST	ADABOOST	KNN	TREE
Abit underfit	ok	Abit overfit but Good	Good target pred	Over fit	well	overfit	Bad target predict

2. Default Credit Card Client

model	acc	w.f1	w.rec	clas.rec	tr.acc	tr.w.f1	tr.w.rec	tr.clas.rec
forest	0.823	0.801776	0.823	0.360062	0.999583	0.999583	0.999583	0.999625
adaboost	0.8295	0.804166	0.8295	0.337702	0.817125	0.790857	0.817125	0.325716
svc	0.7865	0.788587	0.7865	0.53431	0.784833	0.786542	0.784833	0.539427
logistic_r	0.690333	0.714158	0.690333	0.640709	0.689458	0.712131	0.689458	0.642255
tree	0.783167	0.784837	0.783167	0.520432	0.779375	0.780281	0.779375	0.515827
nn	0.817833	0.794492	0.817833	0.336931	0.830375	0.809459	0.830375	0.382094
gnb	0.686333	0.711798	0.686333	0.678489	0.687917	0.711871	0.687917	0.679341
knn	0.812333	0.786801	0.812333	0.313801	0.999667	0.999667	0.999667	0.998502

FOREST	ADABOOST	SVC	Logistic	TREE	NN	GNB	KNN

Bad target predict	Bad target predic	Not good target predict	Okay target	Not good target predict	Bad target predic	Better target predict	Not good target predict (affected by data)

3. Breast Cancer Wisconsin

model	acc	w.f1	w.rec	clas.rec	tr.acc	tr.w.f1	tr.w.rec	tr.clas.rec
tree	0.842857	0.841698	0.842857	0.763636	0.953488	0.953293	0.953488	0.913978
knn	0.921429	0.921766	0.921429	0.927273	1	1	1	1
forest	0.842857	0.841698	0.842857	0.763636	0.953488	0.953293	0.953488	0.913978
adaboost	0.942857	0.94266	0.942857	0.909091	0.991055	0.991037	0.991055	0.978495
gnb	0.871429	0.872917	0.871429	0.963636	0.887299	0.88919	0.887299	0.913978
nn	0.921429	0.921766	0.921429	0.927273	0.951699	0.951857	0.951699	0.94086
logistic_r	0.892857	0.894021	0.892857	0.963636	0.874776	0.876718	0.874776	0.887097
svc	0.9	0.900757	0.9	0.927273	0.996422	0.996427	0.996422	1

TREE	KNN	FOREST	ADABOOST	GNB	NN	Logistic	SVC
Okay target predict	Possibility of overfit	Okay target predict	Good	Very Good	Good	Slightly better than GNB	Good

4. Statlog German Credit Data

model	acc	w.f1	w.rec	clas.rec	tr.acc	tr.w.f1	tr.w.rec	tr.clas.rec
adaboost	0.775	0.774416	0.775	0.603448	0.82375	0.819002	0.82375	0.628099
knn	0.72	0.67275	0.72	0.206897	1	1	1	1
nn	0.7	0.70533	0.7	0.551724	1	1	1	1
svc	0.715	0.720596	0.715	0.586207	0.92	0.91816	0.92	0.797521
gnb	0.71	0.718097	0.71	0.62069	0.7375	0.742428	0.7375	0.644628
forest	0.745	0.728021	0.745	0.396552	1	1	1	1
logistic_r	0.7	0.712349	0.7	0.706897	0.7475	0.755659	0.7475	0.747934
tree	0.595	0.613472	0.595	0.689655	0.715	0.726549	0.715	0.822314

Models and Accuracy

ADABOOST	KNN	NN	SVC	GNB	FOREST	Logistic	TREE
Good	Overfit, Bad target predict	Overfit, Bad target predict	Overfit Bad target predict	Abit underfit but Okay	Over fit Bad target predict	Abit underfit But Good	Low acc

5. Adult

model	acc	w.f1	w.rec	clas.rec	tr.acc	tr.w.f1	tr.w.rec	tr.clas.rec
logistic_r	0.823557	0.807467	0.823557	0.437299	0.824977	0.810051	0.824977	0.45307
random_forest_r	0.857033	0.851861	0.857033	0.609646	1	1	1	1
svc	0.848741	0.840386	0.848741	0.553055	0.855958	0.84849	0.855958	0.577474
nn	0.851198	0.844136	0.851198	0.573633	0.867399	0.861784	0.867399	0.620585
gnb	0.800061	0.772305	0.800061	0.320257	0.801405	0.774781	0.801405	0.332962
knn	0.836149	0.829239	0.836149	0.553055	1	1	1	1
adaboost	0.865786	0.861302	0.865786	0.634084	0.870508	0.866425	0.870508	0.650652
tree	0.849048	0.836006	0.849048	0.498392	0.846783	0.833597	0.846783	0.496977

Logistic	FOREST	SVC	NN	GNB	KNN	ADABOOST	TREE
Bad Target class predict	Overfit but still considerably doing well	Bad target class predict	Bad target class predict	Bad target class predict	Bad target class predict	Godd	Bad target class predict

6. Yeast

model	acc	w.f1	w.rec	tr.acc	tr.w.f1	tr.w.rec
logistic_r	0.606061	0.591098	0.606061	0.602867	0.593673	0.602867
random_forest_r	0.639731	0.624157	0.639731	1	1	1
tree	0.565657	0.528619	0.565657	0.608769	0.590015	0.608769
knn	0.599327	0.582725	0.599327	1	1	1
adaboost	0.447811	0.422872	0.447811	0.48145	0.475344	0.48145
nn	0.555556	0.551593	0.555556	0.91484	0.913636	0.91484
svc	0.579125	0.55965	0.579125	0.639123	0.632403	0.639123
gnb	0.127946	0.119014	0.127946	0.120573	0.115047	0.120573

Logistic	FOREST	TREE	KNN	ADABOOST	NN	SVC	GNB
Under fit but fine	Overfit but fine	Still not good	overfit	Bad	Overfit	Bad	Bad

7. Thoracic surgery data

model	acc	w.f1	w.rec	clas.rec	tr.acc	tr.w.f1	tr.w.rec	tr.clas.rec
nn	0.787234	0.712259	0.787234	0	0.968085	0.96663	0.968085	0.788462
random_forest_r	0.787234	0.712259	0.787234	0	0.981383	0.981868	0.981383	1
tree	0.691489	0.671522	0.691489	0.0555556	0.800532	0.815858	0.800532	0.519231
adaboost	0.797872	0.717613	0.797872	0	0.864362	0.817039	0.864362	0.0769231
svc	0.670213	0.666579	0.670213	0.111111	0.890957	0.902691	0.890957	1
knn	0.808511	0.722904	0.808511	0	0.861702	0.79769	0.861702	0
gnb	0.202128	0.100668	0.202128	0.944444	0.180851	0.116012	0.180851	1
logistic_r	0.638298	0.672668	0.638298	0.5	0.609043	0.669545	0.609043	0.634615

NN	FOREST	TREE	ADABOOST	SVC	KNN	GNB	Logistic
Bad Target predict	Bad Target predict	Better than the others	Bad Target predict	Bad Target predict	Bad Target predict	Lots of misclassification as target class	Better compared to others

8. Seismic bumps

model	acc	w.f1	w.rec	clas.rec	tr.acc	tr.w.f1	tr.w.rec	tr.clas.rec
svc	0.856867	0.880166	0.856867	0.185185	0.935172	0.943019	0.935172	0.902098
tree	0.802708	0.853178	0.802708	0.62963	0.810353	0.849245	0.810353	0.573427
nn	0.938104	0.920519	0.938104	0.037037	0.952588	0.942245	0.952588	0.342657
random_forest_r	0.943907	0.926648	0.943907	0.0740741	1	1	1	1
adaboost	0.945841	0.921396	0.945841	0	0.931301	0.898652	0.931301	0.00699301
knn	0.947776	0.922364	0.947776	0	1	1	1	1
gnb	0.84913	0.880743	0.84913	0.444444	0.855346	0.876291	0.855346	0.440559
logistic_r	0.750484	0.818298	0.750484	0.62963	0.761974	0.817696	0.761974	0.692308

GNB	Logistic	NN	SVC	FOREST	ADABOOST	KNN	TREE
Good	Best	Ok	Ok	Poor	Poor	Worst	Good

Regressions Results

1.1 Red wine

Model Name	MSE	r2 Score	MSE (train)	r2 Score (train)	RMSE
adaboost	0.422842	0.293956	0.33929	0.486776	0.650263
decision_tree_r	0.437467	0.269536	0.378562	0.427373	0.661413
random_forest_r	0.335154	0.440374	0.0459147	0.930548	0.578925
knn_r	0.35835	0.401642	0	1	0.598623
nn_r	0.3883	0.351633	0.324821	0.508663	0.623137
svr	0.384829	0.357429	0.295103	0.553616	0.620346
GaussianProcessRegressor	0.389349	0.34988	0.422238	0.361307	0.623979
linear_r	0.389247	0.350051	0.422265	0.361266	0.623897

GNB	Linear	NN	SVR	FOREST	ADABOOST	KNN	TREE
Ok	Ok	Ok	Ok	Best	Second worst	overfit	Worst

1.2 White wine

Model Name	MSE	r2 Score	MSE (train)	r2 Score (train)	RMSE
random_forest_r	0.450213	0.499261	0.048403	0.936626	0.670979
svr	0.495425	0.448974	0.0472239	0.93817	0.703864
adaboost	0.620828	0.309497	0.499519	0.34598	0.787926
GaussianProcessRegressor	0.662478	0.263173	0.546916	0.283922	0.813927
linear_r	0.662523	0.263123	0.546927	0.283908	0.813955
nn_r	0.515604	0.426531	0.358769	0.530264	0.718056
knn_r	0.465507	0.482249	0	1	0.682281
decision_tree_r	0.663371	0.26218	0.473119	0.380545	0.814476

GNB	Linear	NN	SVR	FOREST	ADABOOST	KNN	TREE
Second worst	poor	Ok	Good	Best	poor	overfit	worst

2. Communities and crime

Model Name	MSE	r2 Score	MSE (train)	r2 Score (train)	RMSE
decision_tree_r	0.0243829	0.540494	0.0196891	0.638483	0.15615
linear_r	0.0181617	0.657734	0.0159471	0.707191	0.134765
adaboost_r	0.0203629	0.616252	0.0178089	0.673005	0.142699
knn_r	0.0217617	0.589891	0	1	0.147519
svr	0.0501837	0.054266	0.00815885	0.850193	0.224017
random_forest_r	0.0184094	0.653068	0.00260516	0.952166	0.135681
nn_r	0.0184407	0.652477	0.0153116	0.718858	0.135797
GaussianProcessRegressor	0.0217053	0.590955	1.10113e-18	1	0.147327

GNB	Linear	NN	SVR	FOREST	ADABOOST	KNN	TREE
Poor	Best	Good	Extreme overfit	Good	Ok	overfit	Poor

3. QSAR aquatic

Model Name	MSE	r2 Score	MSE (train)	r2 Score (train)	RMSE
decision_tree_r	1.70203	0.303	1.06304	0.624037	1.30462
random_forest_r	1.00271	0.58938	0.206493	0.92697	1.00135
GaussianProcessRegressor	1.03838	0.574771	1.5001	0.469464	1.01901
adaboost	1.2229	0.49921	1.13225	0.599559	1.10585
knn_r	0.97883	0.599158	0.00310942	0.9989	0.989359
nn_r	0.92288	0.622071	0.790441	0.720446	0.960666
linear_r	1.03838	0.574774	1.5001	0.469464	1.01901
svr	0.8162	0.665757	0.968502	0.657472	0.903438

GNB	Linear	NN	SVR	FOREST	ADABOOST	KNN	TREE
bad	bad	poor	Best	bad	Bad	poor	Worst

4. Facebook Metrics

Model Name	MSE	r2 Score	MSE (train)	r2 Score (train)	RMSE
adaboost_r	249287	0.52471	774.132	0.989971	499.286
random_forest_r	261335	0.501739	165.39	0.997857	511.209
linear_r	1.59547e-24	1	1.62785e-25	1	1.26312e-12
decision_tree_r	273829	0.477918	1409.11	0.981744	523.287
GaussianProcessRegressor	5390.9	0.989722	6.62958e-06	1	73.4227
knn_r	268191	0.488667	0	1	517.872
svr	0.0291696	1	0.00212006	1	0.170791
nn_r	15.5622	0.99997	2.21569	0.999971	3.94489

GNB	Linear	NN	SVR	FOREST	ADABOOST	KNN	TREE
bad	best	poor	Second best	bad	bad	bad	bad

5. Bike Sharing

6. Student Performance

Model Name	MSE	r2 Score	MSE (train)	r2 Score (train)	RMSE
knn_r	2.89144	0.623864	3.20357	0.706117	1.70042
svr	0.892602	0.883885	1.70198	0.843867	0.944776
GaussianProcessRegressor	1.06664	0.861246	1.56657	0.856289	1.03278
random_forest_r	1.11196	0.855349	1.47377	0.864802	1.0545
decision_tree_r	1.4509	0.811258	1.74272	0.84013	1.20453
adaboost	1.19939	0.843976	1.41953	0.869778	1.09517
linear_r	1.06657	0.861254	1.56657	0.856289	1.03275
nn_r	1.0665	0.861263	1.56657	0.856289	1.03271

GNB	Linear	NN	SVR	FOREST	ADABOOST	KNN	TREE
Good	Good	Good	Best	Ok	Ok	Worst	Poor

7. Concrete compressive strength

Model Name	MSE	r2 Score	MSE (train)	r2 Score (train)	RMSE
linear_r	95.7893	0.63683	110.23	0.609268	9.7872
adaboost	54.7221	0.79253	45.4812	0.838782	7.39744
nn_r	26.7776	0.898477	12.8303	0.954521	5.17471
decision_tree_r	47.9659	0.818145	2.85953	0.989864	6.92574
GaussianProcessRegressor	95.789	0.636831	110.227	0.609276	9.78718
knn_r	66.4848	0.747934	1.27215	0.995491	8.15382
random_forest_r	22.1482	0.916028	4.4845	0.984104	4.70619
svr	49.1082	0.813814	39.8412	0.858775	7.00772

GNB	Linear	NN	SVR	FOREST	ADABOOST	KNN	TREE
worst	worst	Second best	poor	best	poor	bad	ok

8. SGEMM GPU kernel performance