Enhancing Performance in Sensorless Drive Diagnosi Dataset via Feature-Reduction and Instance-Reduction Techniques

Atul Kumar
110143931
atulkum@uwindsor.ca

Nehme Haidura
110203614
haidura@uwindsor.ca

ABSTRACT

This study presents a comparative study of seven machine learning classifiers for Sensorless Drive Diagnosis. We tested linear models (Logistic Regression, SGD Classifier), Tree-based methods (Decision Tree), instance-based learning (KNeighbors Classifier), generative models (Gaussian Naive Bayes), and discriminative classifiers (LDA, QDA). We examined the effects of features-reduction techniques such as feature selection (Recursive Feature Elimination, SelectKBest) and dimensionality reduction (PCA). We also leveraged instance-reduction techniques (Nearest Neighbors, Voronoi Neighbors), and conducted a comparative study against the full dataset. Our study shows that KNeighbors with Recursive Feature Elimination outperformed with the highest accuracy of 99.92%, surpassing the model's performance trained on full dataset.

1 INTRODUCTION

Electrical motor fault diagnosis is critical for maintaining industrial efficiency and preventing costly equipment failures. This study presents a comprehensive comparative analysis of machine learning approaches for Sensorless Drive Diagnosis, evaluating how different classification algorithms perform when detecting various operating conditions in electrical motors using only electric current drive signals. Traditional motor fault diagnosis relies on manual inspection and expert knowledge, which can be expensive, time-consuming, and subject to human error. This is where we need Machine learning since it offers a promising alternative by automatically identifying fault patterns from electrical signal data. The UCI Machine Learning Repository's Sensorless Drive Diagnosis Dataset, containing 58,509 instances with 48 features across 11 operating conditions, provides a strong reference point for evaluating classification algorithms in this domain.

Our study aims to identify the best approach for Sensorless Drive Diagnosis dataset by evaluating seven different machine learning classifiers and observe the impact of feature-reduction and instance-reduction methods compared to the full dataset.

Conference'17, July 2017, Washington, DC, USA
© 2025 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 DATASET DESCRIPTION

The dataset used in this project is the Sensorless Drive Diagnosis Dataset [3] provided by the UCI Machine Learning Repository. The data is real-world data collected from an electrical motor operating under various conditions to detect mechanical faults in it. The dataset is made up of 58,509 instances, where each instance represents a snapshot of the motor's behavior, and 48 features, which are extracted from electric current drive signals. The data is numerical, and there are no missing values, which makes this dataset perfect for classification problems. In this dataset, the 11 classes correspond to 11 different operating conditions of an electromechanical drive system. These 11 different conditions (or classes) represent the state of the drive system, whether the system is functioning normally or experiencing a specific type of failure.

To gain initial insights from the dataset, we projected the 48-dimensional feature space to a 2-dimensional space for visualization. For that, we have utilized a very common dimensionality reduction technique: Principal Component Analysis (PCA). PCA is a linear projection, and it reduces the feature space by identifying the direction of maximum variance. We leveraged this technique to understand the underlying data structure and get some idea of how different models on this dataset could behave.

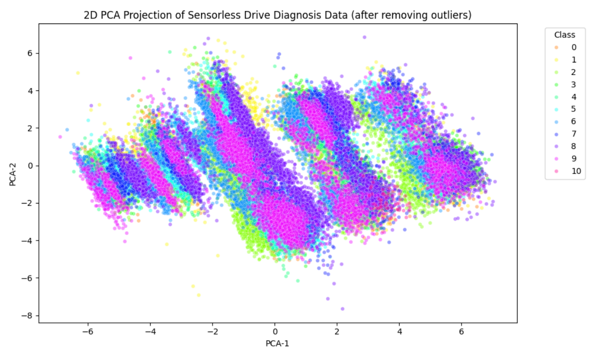


Figure 1: PCA Reduction

As shown in Figure 1, dimensionality reduction with PCA results in a high degree of overlap between many classes, creating elongated and entangled clusters. This can pose several challenges for linear models to classify on the reduced dataset (as we will also see in section 5.3, when dealing with the PCA-reduced data for removing Voronoi Neighbors).

3 RESEARCH QUESTIONS

For this research, we focused on how different machine learning models and data reduction techniques can impact the classification performance on Sensorless Drive Diagnosis dataset. The following are the questions that we formulated:

(1) How do different machine learning models behave on the Sensorless Drive Diagnosis dataset, and which model could contribute best to the performance gains based on the nature of this dataset?
(2) Can techniques like feature selection or dimensionality reduction improve the model performance and make it efficient without sacrificing much of the accuracy?
(3) What happens if we try to reduce the number of training examples using the technique, we studied in class, like Voronoi Neighbors? Would it improve, maintain, or degrade the baseline (full dataset) results?
(4) What model or a combination of models and reduction techniques could offer the best results in terms of performance and complexity for this dataset?

4 METHODOLOGY

4.1 Preprocessing and Reduction Techniques

Before running any experiments, we applied feature scaling to the entire dataset such that each feature has zero mean and a unit variance using scikit-learn's StandardScaler. We conducted extensive experiments on this feature-scaled dataset using techniques learned throughout the course (with a PCA yet to be discussed, but learned in another course). For our analysis, we created the following three different versions of this dataset and performed the same experiments for each of them to do a comparative analysis:

(1) **Full (Original) Dataset:** Our first set of experiments used the entire feature-scaled dataset, while keeping all the features and instances. Later, we use the results from this set of experiments as a baseline while comparing them with the other two versions of the dataset.
(2) **Feature-Reduction Dataset:** To investigate whether a small subset of features could improve, maintain, or even degrade performance, we reduced the number of features using two techniques:
   (a) **Feature Selection:** It involves selecting the relevant features while filtering out the unimportant ones. For selection, we have leveraged the following two methods:
      (i) Recursive Feature Elimination (RFE): It selects the features greedily by recursively removing the least important ones, and here we combined RFE with the Logistic Regression estimator.
      (ii) SelectKBest (SKB): It ranks features based on univariate statistical tests and removes the least important features. We tested this with two scoring functions $f\_classif$ (for continuous features) and $mutual\_info\_classif$ (for non-linear relationships).
   (b) **Dimensionality Reduction:** Alongside feature selection, we also applied PCA, which projects the high-dimensional space to a lower-dimensional space that captures the majority of the variance.

   To create the feature-reduced dataset, we picked 20 features after finding out that PCA were able to capture over 95% of the variance, which is also considered a commonly accepted threshold as suggested in [2]. To have a fair comparison with feature selection approaches, we ended up deciding to extract the top 20 relevant features through feature selection techniques as well.
(3) **Instance-Reduction Dataset:** Lastly, we explored shrinking the datasets in terms of samples, while keeping the features intact. For this, we leveraged two techniques:
   (a) **Nearest Neighbors:** This is an unsupervised nearest neighbor search approach, which searches for a predefined number of closest data points in the neighborhood based on Euclidean distance, by default. For example, when k=3, we identify the 3 nearest neighbors. If all three neighbors belong to the same class as the considered data point, then they are considered redundant and removed from the dataset. We conducted the experiments with k=3, 5, and 7, and sizes of datasets reduced to 20620, 26670, and 30217, respectively.
   (b) **Voronoi Neighbors:** We also reduced the dataset using Voronoi neighbors, where we consider the data point as redundant if it's surrounded by the data points of the same class. However, for this, we had to reduce the 48-dimensional feature space to only 3 dimensions, as Voronoi is only feasible for 2D or 3D feature spaces. To reduce the dimensions, we again leveraged PCA. After this approach, the size of the reduced dataset was 33706.

4.2 Model Selection

To identify the best approach for our dataset, seven different models were selected based on what we have covered in this course and their proven effectiveness in classification tasks, and are as follows:

(1) **Logistic Regression:** Logistic Regression is one of the easiest models to implement and understand. It is fast at classification, and has good accuracy for many simple data sets, but this models is too simple for more complex problems. It was selected for its natural probabilistic view of class predictions.
(2) **SGD Classifier:** The second linear Classifier used in this dataset is the SGD (Stochastic Gradient Descent). This model uses stochastic gradient descent to optimize the model's loss function. It updates the model's parameters gradually with each training sample to make it more efficient for larger datasets.
(3) **Decision Tree Classifier:** The Decision Tree classifier recursively splits a feature space to minimize class impurity at each level. It splits based on the splitting criterion we chose. Here we experimented with two popular splitting criteria:
   (a) Entropy [4]: It measures the amount of uncertainty. The less the uncertainty, the more information gained.
   (b) Gini [4]: It measures impurity as calculating the probability of how often the sample would be mislabeled, if labeled randomly based on the class distribution of the node.

   Both split the feature space such that class impurity improves at each node, while increasing homogeneous regions.
(4) **KNN:** KNN forms complex decision boundaries naturally and adapts to data density. It works well where there are lots of samples, but it faces some problems since it's sensitive to class noise and to the scale of attributes. KNN is expensive a test time because when the model wants to find the nearest

neighbor of a query point x, it must compute the distance to all N training examples.

(5) **GaussianNB**: is a probabilistic classifier which assumes each feature is independent of any other feature, and also that each continuous feature is assumed to be normally distributed within each class.
(6) **Linear Discriminant Analysis (LDA)**: LDA works by separating multiple classes usually two or more to find the linear combination of features. LDA can outperform other classifiers if the data is normally distributed, and it helps in removing noise. LDA assumes all classes share the same covariance matrix.
(7) **Quadratic Discriminant Analysis (QDA)**: This machine learning technique for classification problems. It's more flexible than LDA and unlike it, QDA allows each class to have its own covariance matrix.

In total, all mentioned ML models were tested on the full dataset. Based on accuracies, the top three were selected for further evaluation on feature-reduced and instance-reduced datasets, due to space constraints.

4.3 Hyperparameter Tuning

This process is to find the best set of hyperparameters for ML model. It is also known as hyperparameter optimization. To achieve optimal performance, some models need hyperparameter tuning, which is set before the training process starts. The right combination can help increase the accuracy and decrease the training time.

Based on each model and its characteristics, different hyperparameters were selected to get the models to perform their best. A sklearn's GridSearchCV was applied, and this allowed us to evaluate each combination on the validation set and observe the performance.

For logistic regression, we tested regularization strength (c) and solvers such as $lbfgs$ and $saga$.

For Decision Tree, we tuned max_depth, min_samples_split and min_samples_leaf to prevent overfitting. Finally, we compared 'gini' and 'entropy' criteria.

For the SGD Classifier, we explored different loss functions ('hinge' and 'log_loss'), penalties ($L1$, $L2$), learning rate ($optimal$, $invscaling$), and $alpha$.

KNN tuning was simply to find k, while comparing Manhattan and Euclidean distances.

Gaussian Naive Bayes tuning included var_smoothing parameter to help improve the predictive performance.

For LDA, we used shrinkage and solver combinations. While in QDA, reg_params were used to regularize and prevent overfitting.

4.4 Training and Evaluation

To train and evaluate our data, we split our dataset in a ratio of 80%:20%, where 80% is for training and 20% is for the test set. During the training phase, each model was evaluated using a consistent set of hyperparameters to ensure fair experiments, and also on the same dataset variant (for example, full, feature-reduced, etc.). We performed a grid search on the specified set of parameters using 3-fold cross-validation to test the generalization of the models with each of the hyperparameter configurations and find the best set of them for each model. As our research contains several experiments, that's why, we were very selective to choose the metric(s) to evaluate. Since our dataset was fully balanced for all the classes, we reported $Accuracy$ and $F1Score$ (for training and validation sets), as close similarity between $Accuracy$ and $F1Score$ metrics in this case, suggests that $Precision$ and $recall$ are balanced, too. For the reduced datasets variants, we reported accuracies of each model across all types of reduced datasets.

Note: The highest validation accuracies for each model (or a combination of model and data reduction) are marked with a large cyan-colored dot in all plots for easy identification.

5 RESULTS AND DISCUSSIONS

5.1 Full Dataset

(1) **Logistic Regression:**

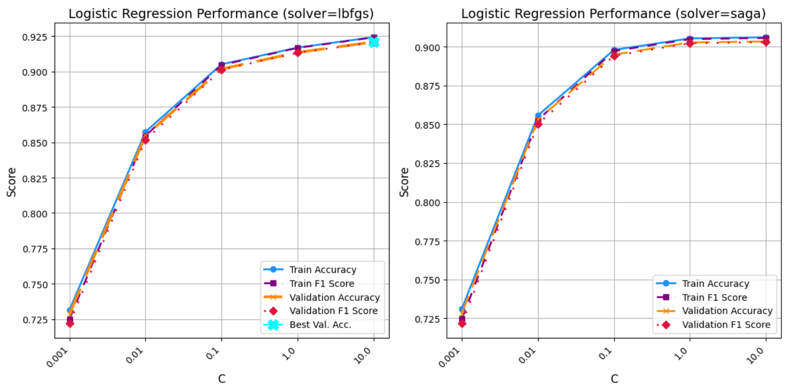


Figure 2: Logistic Regression: $lfbgs$ (left) vs $saga$ (right)

As shown in Figure 2, we have tested LogisticRegression with two different optimization approaches: $lbfgs$ (leverage hessian approximation) and $saga$ (stochastic gradient descent-based solver performs incremental updates), and a set of regularized parameters c. It is clear in Figure 2 that both solvers yield consistent performance at particular values of c, demonstrating that both solvers converge to similar minima of the convex loss function. However, the regularization parameter (c) demonstrated a significant impact on the overall performance. As we know, when c is small, strong regularization is applied, and it keeps the model weights small, which results in reducing overall complexity and preventing overfitting of the model. However, it also underfits the complex patterns in the data, resulting in lower performance. This is why, as we increased c, the model allowed more flexibility in assigning larger weights to features, capturing more class-specific patterns, and resulting in improved performance. However, when we increased c beyond 0.1, performance gains became marginal. It's probably because the model had already captured most of the useful features at 0.1, and increasing further could result in overfitting.

(2) **SGD Classifier:** Depending on the combinations of hyperparameters, the overall performance of the model varied considerably. We have used combinations of loss functions (hinge, log_loss), penalty (l1, l2, elasticnet), regularization parameter (alpha), and learning rate (optimal, invscaling).

Overall, as shown in Figure 3, *optimal* learning rate (on the left) outperformed *invscaling* (on the right), as it allows the model to converge more efficiently, unlike *invscaling*, which gradually decreases the learning rate over time. Due to this, *invscaling* can lead to slow convergence or even premature stagnation, while *optimal* adapts the learning rate in a more balanced way, helping the model escape the local minima. The *log_loss* yielded stronger results than hinge in both the learning rate strategy, probably because of its probabilistic nature and smooth decision boundary (same as LogisticRegression).

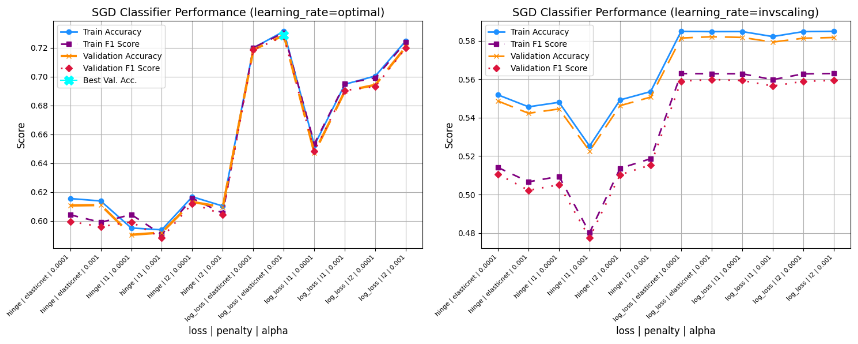


Figure 3: SGD Classifier: *optimal* (left) vs *invscaling* (right)

While learning rate is *optimal*, increasing the regularization has almost no effect when using *hinge* loss, demonstrating that regularization did not significantly impact the decision boundary. However, opposite to LogisticRegression, with an increase in *alpha* (stronger regularization), SGDClassifier showed slight improvements when using *log_loss*. For learning rate *invscaling*, even though the model performed better with *log_loss*, compared to *hinge* loss, however, it has shown little to almost no change with different penalties, which most likely represents the model underfitting.

(3) **Decision Tree:** As we can see in Figure 4, for both splitting criteria (i.e., *entropy* and *gini*), a significant improvement in the performance is witnessed when the depth of the tree is increased (achieving the highest of 98% accuracy). This shows that allowing greater depth has a significant impact, since allowing trees to grow captures the fine-grained, class-specific patterns present in the data. We tested with the depths: 5, 10, 20, *nan*. Here *nan* means that, tree will grow until all nodes are pure (no class-mixing) or the leaf nodes contain fewer samples than *min_samples_leaf*.

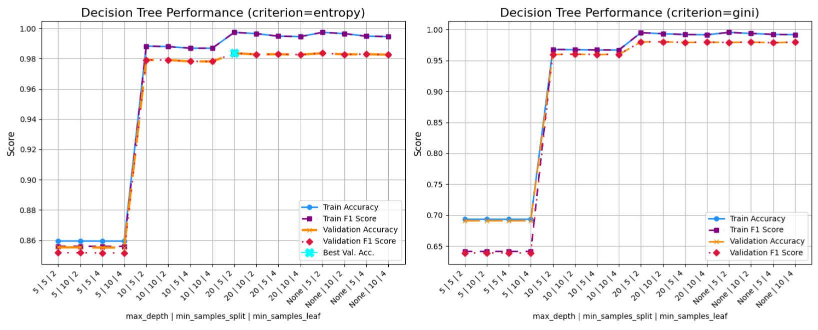


Figure 4: Decision Tree: *entropy* (left) vs *gini* (right)

Out of all, *max_depth* of 5 showed the worst performance, likely underfitting the data and failing to capture complex patterns. However, as we relaxed this constraint, it significantly improved the performance, but marginally lower gains beyond *max_depth* of 10, likely because the model had already captured the complex patterns. It is also important to note, both *min_samples_leaf* and *min_samples_split* has little to no impact on overall performance across the same depths because the dataset probably contained sufficient samples in the node to satisfy both parameters.

With regards to the *criterion* parameter, *entropy* has outperformed *gini* across all the experiments. One of the key reasons behind this may be that when the nodes contain a mix of different classes, entropy makes splits based on the information gain, while *gini* doesn't care about uncertainty but on reducing impurity, probably causing less informative splits [6]. However, the margin of improvement varied with different trees' depths. When tested with the depth of 5, *entropy* outperformed *gini* by a margin of almost 20% on the validation set, likely because each split gets more important with shallow trees and has to make the most out of each level. Whereas, when we increase the depth, both *criterion* get enough chances to refine the partitions, resulting in achieving similar performance.

(4) **K-Neighbors Classifier:** The performance of this model primarily depends on two hyperparameters, i.e., the number of closest neighbors (k) and distance metric (p). For KNeighbors, we performed a wide set of experiments, ranging k from 3 to 15 and leveraging Manhattan (p=1) and Euclidean distance (p=2).

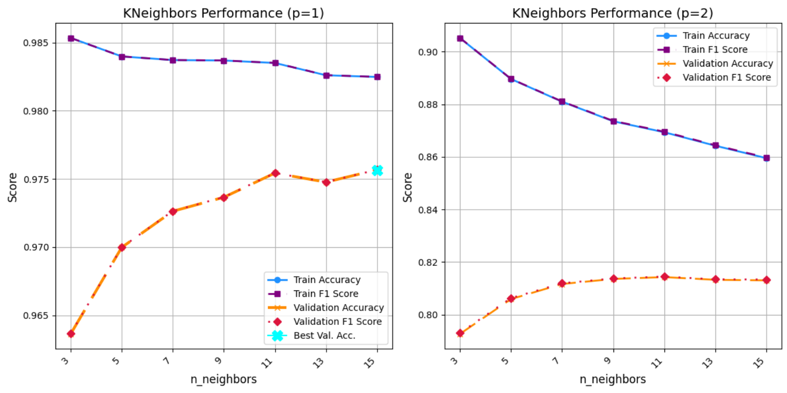


Figure 5: K-Neighbors Classifier: p=1 (left) vs p=2 (right)

Overall, the results were significantly higher when the model computed the nearest neighbors using Manhattan distance in contrast to Euclidean distance. While using Manhattan distance, as we increased the k, the performance of the model improved consistently, reaching over 97.5% with k=15. However, for Euclidean, it increased up to k = 9 (81%) and then appeared slightly dropping for bigger $k's$.

Since selecting k could be tricky and lead to overfitting or underfitting. For example, lower values of k can have high variance (sensitive to individual neighbors), while large k could have high bias (smoothing over a large number of neighbors) [5]. This is why increasing k has improved the performance in Figure 5 for both distance metrics, as the model becomes more robust and less sensitive to individual neighbors. With regards to Manhattan vs Euclidean performance, as mentioned in [1], Manhattan distance is considered to be a more preferable option than Euclidean, especially in high-dimensional space. This also justifies the significant difference in the model's performance with Manhattan and Euclidean distance for our 48-D dataset. Probably the drop that we see beyond $k = 9$ while utilizing Euclidean distance could also be due to this very reason that as we increased k, Euclidean distance may have started including farther and irrelevant neighbors, especially when we have outliers in our dataset.

(5) **Gaussian NB:** As we know, NB expects the data to be normally distributed and uncorrelated. However, our dataset has the readings from the sensors, which can be correlated, and eventually affect the model's performance. For confirmation, we also drew the correlation map between features, and some of them seemed to be highly correlated. Also, we checked the distribution for each feature with each class to confirm the feature distribution and realized the majority of them were not normally distributed (either skewed or having multiple peaks). Given the high dimensionality of our dataset, we chose not to include the correlation map and feature distribution here.

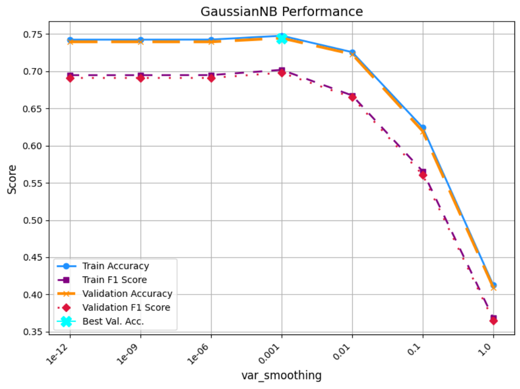
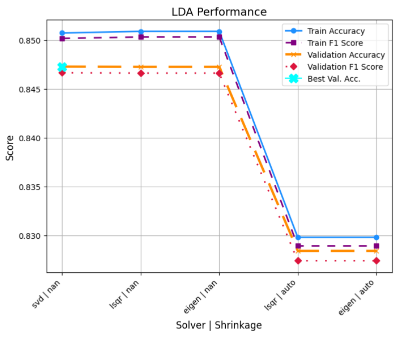


Figure 6: Gaussian NB

Since our dataset violates these two assumptions, we expected the model performance not to be as high (reaching the highest of 72%) as LR or KNN. For this model, we experimented with *var_smoothing* hyperparameter, which essentially adds a predefined value to the variance of the distribution, resulting in smoothing a curve. This helps include samples in the tails of the distribution for more coverage.
When experimented with different values, the model performed almost the same by adding a small variance to the distribution (achieving around 75% accuracy). However, when we went beyond 0.001, the model started performing worse with each increase. This is most likely because it has significantly widened the tails of the distribution, making each class look almost identical, severely affecting the overall performance.

(6) **LDA & QDA Classifier:** We tested LDA with different solvers such as *svd*, *lsqr*, *eigen*, with shrinkage of *none* and *auto*. *lsqr*, *eigen* compute the covariance matrix in different ways, whereas *svd* don't compute compute the covariance. Whereas, in contrast to *none* shrinkage, *auto* regularizes the covariance matrix.



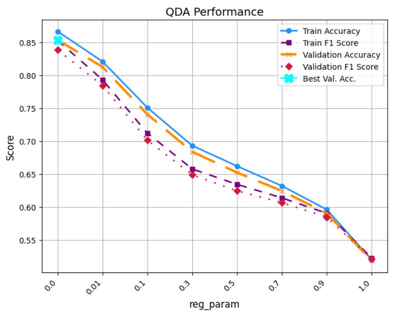


Figure 7: Discriminative Classifiers: LDA (left) vs QDA (right)

Since LDA also expects the data to be normally distributed, which isn't in our case, resulting in the highest accuracy of almost only 85%, as shown in the left plot of Figure 7. All the solvers performed the same when using the same shrinkage parameter. However, when we applied regularization to the covariance matrix, a sudden drop was witnessed. Since shrinkage shrinks the covariance matrix estimation towards a diagonal matrix, and over-regularizes the well-estimated matrix, probably leading to underfitting. We also suspect that resulted lower accuracy could also be due to heavy overlap, when we project into a lower-dimensional space, as we saw in the case of PCA (Figure 1)
QDA, as the name suggests, quadratic, this model is capable of generating non-linear decision boundaries. However, it also considers the normal distribution within each class. Surprisingly, QDA performance was not much different from LDA, as shown in the right plot of Figure 7, even though it is capable of drawing non-linear decision boundaries. It has achieved a maximum of 85% accuracy across different values of hyperparameter *reg_param* (which again, regularizes the covariance matrix as LDA). Since, while projecting, both maximize the between-class separation, which might naturally be separated by linear decision boundaries to a great extent. Similar to LDA, when we applied the regularization to the covariance matrices, the performance starts to decline, suggesting over-regularizing the matrices.

5.2 Feature-Reduction Dataset

(1) **Logistic Regression:** Similar to the full dataset study we tested LogisticRegression with $lbfgs$ and *saga*, and a set of regularized parameters c. But at the same time we explored the impact of 4 different feature extraction techniques. The four approaches are: *RFE_LR* (Recursive Feature Elimination with Logistic Regression), *SelectKBest* with both classification and mutual information scoring functions, and PCA (Principal Component Analysis). As shown in figure 8, all the methods have nearly similar performance while c increases

from 0.001 to 0.1, then it stabilizes around 0.90 accuracy with both *lbfgs*, *saga* solvers, with the *SelectKBest_Classif* method achieving the highest score in both solvers as well. Our experiment shows that both solvers produce nearly identical results regardless of which feature extraction method is used, which we can conclude that the optimization algorithm doesn't have as much impact as the quality of the features and the strength of regularization applied.

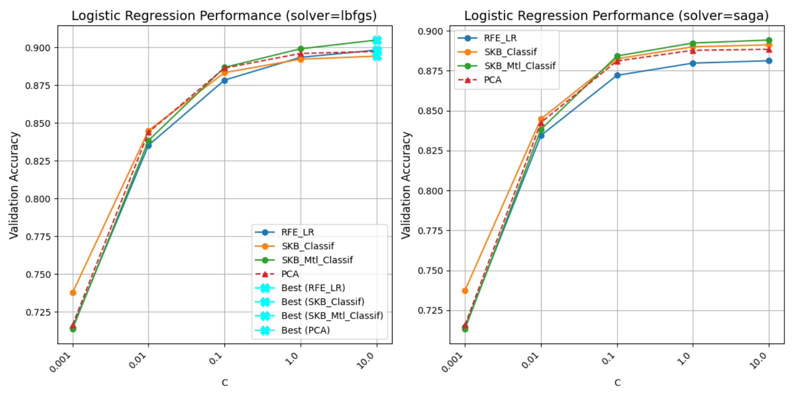


Figure 8: Logistic Regression: *lfbgs* (left) vs *saga* (right)

(2) **Decision Tree:** As in the first study we made on the full dataset, we experimented here with two popular splitting criteria, *Entropy* and *Gini*. As shown in Figure 9 at depth 5, all feature extraction methods perform poorly, but once we allow deeper trees (depth 10 and beyond), performance increases to reach nearly perfect results for most methods. This sudden improvement suggests that the data contains complex, non-linear patterns. It is worth noting that among

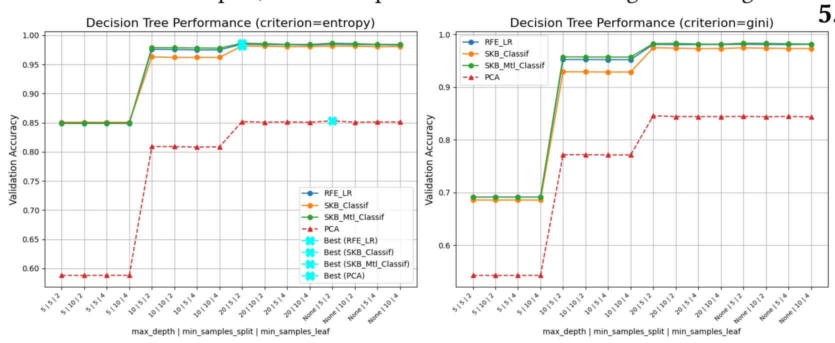


Figure 9: Decision Tree: *entropy* (left) vs *gini* (right)

all the feature only *PCA* don't have exceptional performance with deeper trees. While it also increases when the depth increase but it stables at 85% accuracy even with the unlimited tree depth(nan), this shows that *PCA* is fundamentally removing class-relevant information that even a highly flexible decision tree cannot recover. Worth noting that the increase from 85% and 70% for *Entropy* and *Gini* respectively to 90% between depths 5 and 10 indicates that the data contains complex patterns requiring deeper trees to capture

(3) **K-Neighbors Classifier:** For the KNeighbors classifier, we repeated the same experiment as in the first study, but adding our feature selection and dimensionality Reduction methods at the same time. As shown in Figure 10, Manhattan distance delivers exceptional results for most methods, achieving a perfect accuracy rate between 99% and 100% that remains stable across all k values, while Euclidean distance shows notably weaker performance. The supervised feature selection methods outperform *PCA* in both Manhattan and Euclidean distance, this underperformance of *PCA* could be because it limits the ability of KNN to make accurate predictions through neighborhood voting. The stability performance

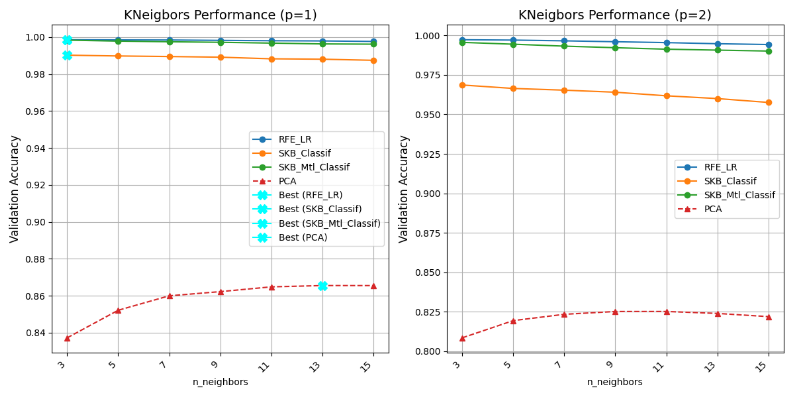


Figure 10: K-Neighbors Classifier: p=1 (left) vs p=2 (right)

while increasing k values for Manhattan distance confirms that Manhattan distance maintains meaningful neighborhood relationships in our feature space, while Euclidean distance shows slight degradation, which indicates that it increasingly includes irrelevant and distant points that hurt classification accuracy.

5.3 Instance-Reduction Dataset

(1) **Logistic Regression:** When using Nearest Neighbors on the full 48-dimensional feature space, the performance improved significantly as we increased C. As we mentioned, increasing C reduces the regularization, which allows the model to assign higher weights and better fit the training data. And, as we can see in Figure 11, instance reduction with large k increases the performance due to sufficiently broad samples, while making decisions whether the data instance is redundant or not, and leading to less noisy neighborhood estimates.

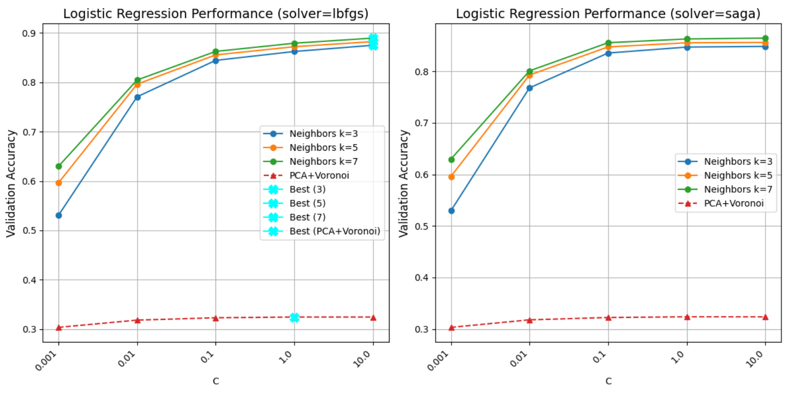


Figure 11: Logistic Regression: *lfbgs* (left) vs *saga* (right)

On the other hand, when we reduce the instances through the PCA+Voronoi approach, the performance becomes significantly worse. This is mainly because PCA compresses the 48-dimensional feature space to just 3 dimensions, which leads to entangled and overlapped clusters, as we also saw in Figure 1. Due to the major loss of information, Voronoi failed to capture meaningful instances. Since the reduced PCA results in significant overlapping between classes as well, logistic regression performed worse. That is expected, as logistic regression is a linear model, which requires the features to be linearly separable. Additionally, both the $lbfgs$ and $saga$ solvers achieved comparable results, with $lbfgs$ being a slightly better overall.
Another important observation is that even after substantial instance reduction in the dataset, the model achieved results that were quite comparable to its baseline counterpart, with only a slight accuracy drop of 2.5%

(2) **Decision Tree:** As we can see in Figure 12, Decision Tree also showed that the model performance could be affected by the instance selection strategy. For the Nearest Neighbors instance reduction, the model was able to achieve near-perfect validation accuracy, similar to the baseline, especially when we allowed the tree to grow for at least a depth of 10. It shows that the remaining instances retained enough structures for the tree to learn effective splits. For the PCA+Voronoi, unlike Logistic Regression, the Decision Tree was able to achieve over 60% accuracy on the validation set, likely because it can capture non-linear decision boundaries, allowing it to extract some useful underlying structure, despite having a low-dimensional space so much entangled and overlapped. Across both types of instance reduction, the choice of impurity measure had little effect.

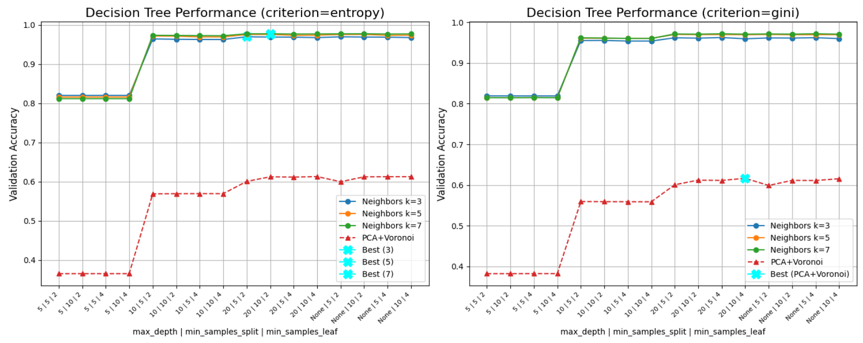


Figure 12: Decision Tree: $entropy$ (left) vs $gini$ (right)

With regards to the baseline results, the model with even less representative data has been able to achieve a maximum accuracy of around 97.75%, compared to the baseline of around 98.36%

(3) **K-Neighbors Classifier:** As shown in Figure 13, KNeighbors results were strongly influenced by the distance metric and structure of the reduced dataset. Using Manhattan distance (p=1), the nearest-neighbors reduction performed very well, all reaching 92%-95% with just k=5. As we increased the k closest neighbors, performance keeps consistently increasing, but marginally, as models get more robust and less sensitive to individual neighbors with large k. While for PCA+Voronoi reductions instances, it performed a little better than even Decision Tree, reaching around 68%. It's likely because KNeighbors relies on local neighborhood information, which might be getting adapted better in our reduction approach, even when class boundaries were not clear.

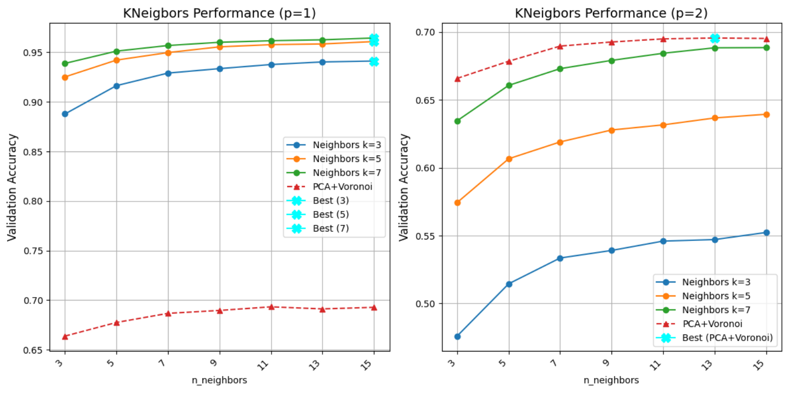


Figure 13: K-Neighbors Classifier: p=1 (left) vs p=2 (right)

Surprisingly, for Euclidean distance (p=2), PCA+Voronoi outperformed the Nearest Neighbors-based reductions, attaining almost 70%. While overall performance was significantly lower on Nearest Neighbor-based reduction methods compared to Manhattan results, PCA+Voronoi showed its best results using Euclidean distance. Since PCA turns the data into lower lower-dimensional space, where the distance between the points is more suitable to be calculated by Euclidean distance. As a result, KNeighbors was able to find the useful neighbors in this space, even though most of the information has been lost during the PCA and instance reduction.
In comparison to baseline which achieved around 97.5% on validation set, we still managed to gain over 95% accuracy with a significant reduction of the dataset.

5.4 Generalization on Test Set

After experimenting with different models, hyperparameter configurations, and data reduction methods (feature-reduction and instance-reduction), we leveraged the best-performing models on the unseen data to evaluate the generalization. To keep the analysis concise and within the report's page limit, we only reported the best-performing reduction technique for each model. For example, for the instance-reduction scenario, we picked NearestNeighbors (k=7), as it outperformed every time compared to other variants such as k=3, k=5, as well as PCA+Voronoi reduction. Similar is the case with the feature-reduction scenario, where we presented the best results reported by any reduction technique (SelectKBest and RFE) on the validation set.
As we can see in Table 1, we evaluated the model on four different metrics i.e., Accuracy, Precision, Recall, and F1 Score. With no reduction applied, models like Decision Tree and KNeighbors outperformed, achieving over 98% across all metrics. While a simple linear model like Logistic Regression performed reasonably well, with over 92% accuracy. Since

Table 1: Test set performance on Original, Feature-Reduced, Instance-Reduced datasets. "-" indicates no reduction applied and models tested on the full dataset. **Bold red** indicates the highest performance, **bold blue** indicates the second highest performance, and **bold orange** indicates the third highest performance.

Dataset	Model	Best Reduction Method (if any)	Generalization on Test Set			
			Accuracy	Precision	Recall	F1 Score
Original (Baseline)	LR	-	92.47%	92.49%	92.47%	92.47%
	SGD	-	74.62%	74.62%	74.62%	74.50%
	DT	-	**98.83**%	**98.83**%	**98.83**%	**98.83**%
	KNN	-	98.21%	98.22%	98.21%	98.21%
	NB	-	73.03%	77.55%	73.03%	67.43%
	LDA	-	84.96%	84.93%	84.96%	84.91%
	QDA	-	83.49%	86.71%	83.49%	81.15%
Feature -Reduced	LR	SKB + mutual_info_classif	90.26%	90.26%	90.26%	90.24%
	DT	SKB + mutual_info_classif	**99.08**%	**99.08**%	**99.08**%	**99.08**%
	KNN	RFE + LogisticRegression	**99.92**%	**99.92**%	**99.92**%	**99.92**%
Instance -Reduced	LR	NearestNeighbors (k=7)	85.81%	88.64%	85.81%	85.21%
	DT	NearestNeighbors (k=7)	98.12%	98.16%	98.12%	98.12%
	KNN	NearestNeighbors (k=7)	85.89%	90.09%	85.89%	82.40%

the data was not normally distributed, leading to only 73% accuracy with Naive Bayes.

Out of four feature-reduction techniques that were implemented, the combination of SelectKBest with $mutual\_info\_classif$ and Recursive Feature Elimination (RFE) with Logistic Regression provided exceptional results. Especially, KNeighbors, when trained on the RFE-reduced dataset, outperformed all the models across all different datasets (i.e., original, feature-reduced, and instance-reduced), achieving the highest performance on this dataset of 99.92%. While Decision Tree using features selected through SelectKBest with mutual_info_classif achieved the second highest performance of over 99% across all metrics.

With regards to instance-reduction, again the Decision Tree with instances selected via NearestNeighbors (k=7) performed exceptionally well. Whereas, both Logisitic Regression and KNeighbors showed the similar performance on the test set, achieving only around 85%. Surprisingly, despite KNeighbors having 96% on the validation and 97% on the training, achieved only 85% indication a overfitting to a training data. Overall, Decision Tree performed consistently well across all the settings. However, KNeighbors with a feature-reduction approach achieved the highest accuracy, indicating the impact of feature selection. Decision Tree with SelectKBest, a feature selection technique, also secured the second-highest score. Even though Decision Tree achieved reasonable results with instance-reduced data, however, it did not rank among the top three, suggesting feature-reduction techniques for this dataset contributed more to the performance of the models than instance-reduced data.

6 CONCLUSION/FUTURE WORK

This study demonstrated that feature selection combined with the right classification algorithms can achieve exceptional performance in Sensorless Drive Diagnosis. Our comprehensive evaluation of seven machine learning classifiers across three datasets (Full, Feature-Reduction, and Instance-Reduction) shows that KNeighbors using the RFE-reduced dataset outperformed with the highest accuracy, surpassing its baseline performance on the full dataset. Moreover, Decision Tree demonstrated the consistent performance across all datasets, implying its robustness across all variants. Overall, our research highlights the importance of feature selection for machine learning tasks, which can not only make the training efficient but also enhance the accuracy and model generalization.

Future research should explore models including Random Forest, ensemble methods, and deep neural networks across all variants of datasets. Additionally, to overcome *PCA* limitations, non-linear dimensionality reduction techniques such as UMAP could be used.

7 STATEMENT OF CONTRIBUTION:

Nehme Haidura: Implemented KNN, GaussianNB, LDA and QDA for full dataset. Implemented the whole pipeline for the future-reduction dataset. Wrote the following sections: Abstract, Introduction, 4.2, 4.3, 5 and 6 **Atul Kumar**: Implemented Logistic Regression, SGD Classifier, Decision Tree for full dataset. Implemented the whole pipeline for instance-reduction experiments. Along with that, worked on writing report for the above mentioned parts as well as Section 2, 3, 4.1, and 4.4

REFERENCES

[1] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. 2001. On the Surprising Behavior of Distance Metrics in High Dimensional Space. In *Database Theory — ICDT 2001*, Jan Van den Bussche and Victor Vianu (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 420–434.

[2] Baeldung. 2024. How Many Principal Components to Take in PCA? (2024). https://www.baeldung.com/cs/pca Accessed: 2025-06-12.

[3] Martyna Bator. 2013. Dataset for Sensorless Drive Diagnosis. UCI Machine Learning Repository. (2013). DOI: https://doi.org/10.24432/C5VP5F.

[4] GeeksforGeeks. 2025. Decision Tree in Machine Learning. https://www.geeksforgeeks.org/decision-tree-introduction-example/. (2025). Accessed: 2025-05-30.

[5] IBM. What is the k-nearest neighbors (KNN) algorithm? https://www.ibm.com/think/topics/knn. (????). Accessed: 2025-06-20.

[6] Stewart Kaplan. 2025. ML 101: Gini Index vs. Entropy for Decision Trees (Python). (2025). https://enjoymachinelearning.com/blog/gini-index-vs-entropy/ Accessed: 2025-06-08.