Academic Integrity Notice: If you reference, adapt, or use any information, analysis, or ideas from this report in your own MECH 501 project, you must provide proper citation to this work. Failure to do so constitutes plagiarism under: Code of Student Conduct and Disciplinary Procedures, Article 16(a).

McGill University

Faculty of Engineering, Department of Mechanical Engineering

MECH 501: Probability, Statistics, and Machine Learning in Mechanical Engineering

Final Project

Bearing Fault Detection and Classification from Vibration Signals

Dr. Audrey Sedal

Cagri Arslan [redacted]

March 30, 2025

Introduction

Rotating machinery, particularly bearings, is widely used in various contexts, environments, and applications. Bearings are present and prevalent in the automotive sector, industrial machinery, and common household appliances, among other industries. Due to the often heavy and repetitive loads they are subject to, unexpected bearing failures can lead to significant downtime and economic losses. Early detection and classification of bearing faults based on vibration data can substantially reduce downtime and maintenance costs by predicting failures before they occur. To address this problem, vibration data from the Case Western Reserve University (CWRU) Bearing Data Center will be analyzed. Specifically, a Hidden Markov Model (HMM), a probabilistic method well-suited for capturing sequential dependencies in vibration signals, will be compared against a simpler probabilistic baseline, the Naïve Bayes classifier, known for its computational speed and interpretability. These methods will be compared to evaluate their effectiveness and practicality in accurately classifying bearing conditions and quantifying uncertainties in fault detection.

The CWRU Bearing Dataset is an open-source dataset provided by the Case School of Engineering of CWRU. It was generated by artificially inducing faults in bearings using electro-discharge machining (EDM). Faults were introduced in the inner raceway (IR), rolling element (B), or outer raceway (OR) with diameters of 0.007", 0.014", 0.021", or 0.028". *Figure 1* illustrates the locations of these faults within a bearing. These bearings were then either put in the drive-end (DE) or fan-end (FE) of a motor, with a healthy bearing placed in the other end, and vibration signals were collected from accelerometers placed at the drive-end and fan-end of the motor. During some experiments, vibration signals were also collected from an accelerometer placed on the base plate supporting the motor.

For OR faults, an additional distinction is made based on their position relative to the load zone, which is centered at 6:00. Accordingly, time-series data can correspond to an OR-centered fault (6:00), an OR-orthogonal fault (3:00), or an OR-opposite fault (12:00). A control set of fault-free bearings was also tested under the same conditions as the faulty bearings.

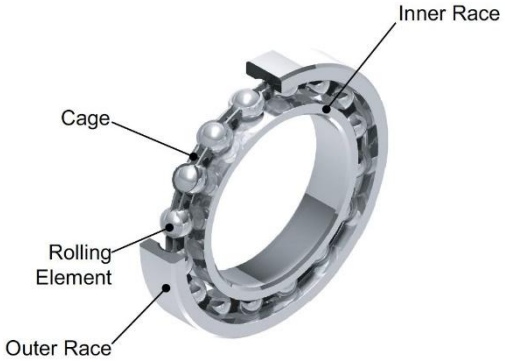

Figure 1: Ball, Inner and Outer Race of a ball bearing

Vibration data was recorded under four load conditions with corresponding rotational speeds: 0 hp (1797 RPM), 1 hp (1772 RPM), 2 hp (1750 RPM), and 3 hp (1730 RPM). Drive-end data was collected at both 12 kHz and 48 kHz, whereas fan-end data was collected only at 12 kHz.

The models developed in this project aim to determine whether a bearing has a fault based on its time-series data and, if so, to classify the type of fault present. A Naïve Bayes model and a Hidden Markov Model will be used due to their interpretability and ability to address multi-class classification problems.

Naïve Bayes is a robust and interpretable probabilistic classifier that efficiently leverages discriminative features, such as time or frequency components from vibration signals, to differentiate between various bearing fault types. Bearing faults often generate distinct frequency

signatures (for example, amplitude spikes at points of impact): characteristics that Naïve Bayes can capture effectively through its assumption of conditional independence among features. This simplicity provides clear probabilistic insights into the classification decision, which is valuable for diagnostic applications where understanding the likelihood of each fault type is informative.

Hidden Markov Models, on the other hand, are well-suited to modeling the temporal dynamics present in vibration data. Vibration signals from bearings exhibit sequential patterns, such as periodic impacts or transient events, that are indicative of different fault conditions. HMMs excel in capturing these temporal dependencies by modeling the transitions between hidden states that correspond to underlying physical processes. They can be adapted to a multi-class classification problem without much difficulty by training one HMM per class, and then applying a maximum likelihood determination to select the class that a given signal is most likely to fall in.

Background

The CWRU bearing dataset is a popular dataset for data analysis and machine learning models due to the well-organized, labeled data within it. Many papers use this dataset as a standard reference for developing new algorithms, yielding impressive performance metrics [2], [3], [4]. However, several recent studies [5], [6] have suggested that the models presented in these papers suffer from data leakage and may not sufficiently generalize to new data due to the procedure used to generate the CWRU bearing dataset.

Of particular significance, the bearings containing different-sized faults were reused to generate data at different speeds [5]. Hence, the bearing containing, for example, a 0.007” IR defect was reused to generate the 0.007” IR 0 hp signal, the 0.007” IR 1 hp signal, the 0.007” IR 2 hp signal, and the 0.007” IR 3 hp signal. This resulted in the same bearing’s vibration signature appearing in different data points. Coupled with the fact that many machine learning models trained on the CWRU bearing dataset are split into train and test sets by motor speed, this results in signals from the same bearing appearing in both the training and testing sets [5]. Hence, the model “learns” or “memorizes” the specific bearing’s vibration signal rather than how to identify the type of fault from the vibration data, returning inflated performance metrics that would fail to generalize to new bearings. This phenomenon is called data leakage. Given that the rotation speeds do not vary significantly (1730 to 1797 RPM), it is argued that despite presenting very high accuracy metrics, models trained on the conventional train-test split will fail to maintain their performance when completely new bearings’ signals are introduced.

Case in point, Hendriks et al attempt to mitigate the occurrence of data leakage by splitting the dataset by fault size, rather than by motor speed. As expected, this results in the previously mentioned CNNs performing significantly worse than with the configuration with data leakage, with accuracies dropping from an average range (between the different CNNs) of 84.9-95.0% to 37.1-53.1% [5]. Abburi et al attempt to remediate this problem by splitting the data into training, testing, and validation sets by both fault size and bearing location, given that different bearings are used on the drive-end and fan-end of the motor [6]. Abburi et al show that, among others, a Naive Bayes classifier with frequency domain data is impressively accurate in the binary (healthy vs faulty) problem, and moderately accurate in the multi-class (healthy, IR, OR, B) problem.

Methodology

This project attempts to build on the work of Hendriks et al and Abburi et al by comparing the efficacy of a Hidden Markov Model and Naïve Bayes multi-class classifier on the CWRU bearing dataset, while minimizing the effects of data leakage by applying similar data partitioning strategies.

For the sake of interpretability, bearing data files will be named and referred to in the following structure:

RPM_Fault_Size_LocationFrequency

RPM: 1730 (3 hp), 1750 (2 hp), 1772 (1 hp), 1797 (0 hp)

Fault: IR, B, OR@6, OR@3, OR@12, Normal

Size: 7 (0.007”), 14 (0.014”), 21 (0.021”), 28 (0.028”)

Location: DE (drive-end), FE (fan-end)

Frequency: 12 (kHz), 48 (kHz)

The initial data files from CWRU are MATLAB (.mat) files and have not been cleaned. However, a public repository exists containing the cleaned data files in NumPy (.npz) format [7]. Data from this repository was checked for errors, and once it was verified that there were no apparent errors, it was used for the purposes of this project.

As only about half of the data files contained time-series data from the base plate, all base plate time series data were discarded. Additionally, fan-end time series data were only absent for the 0.028” faults, so the 0.028” faults were excluded from the study. As a result, we were left with twelve drive-end, fan-end signal pairs per fault and size. For example, for the IR faults, the twelve faults of size 0.007” were:

1730_IR_7_DE48	1750_IR_7_DE48	1772_IR_7_DE48	1797_IR_7_DE48
1730_IR_7_DE12	1750_IR_7_DE12	1772_IR_7_DE12	1797_IR_7_DE12
1730_IR_7_FE12	1750_IR_7_FE12	1772_IR_7_FE12	1797_IR_7_FE12

The OR faults are important to analyze closely as two considerations present itself. Firstly, there are three types of OR faults, as stated in the introduction. Hence, there are more OR faults than other types of faults. If we were to include all OR faults in our data, we risk causing the model to be biased towards the majority (OR) class, and consequently having lower recall metrics for the minority classes. Additionally, a complete basis of data for each OR fault type (OR@6, OR@3, OR@12) are not present, as is the case for the other faults. For example, no samples of 0.014” OR@12 faults exist and the OR@3 files only have FE12 data, not DE12 or DE48. For this reason, only the OR-centered (OR@6) faults were used. For the rare cases where OR@6 data is not present, it was substituted with OR@3 faults. These cases are:

1730_OR@6_14_FE12	1750_OR@6_14_FE12	1772_OR@6_14_FE12
1730_OR@6_21_FE12	1750_OR@6_21_FE12	1772_OR@6_21_FE12

We acknowledge that this is a potential limitation of the models developed (namely with regards to its ability to identify OR@3 and OR@12 faults), but argue that due to the similar nature of the different OR faults and the potential drawback of incorporating incomplete and unbalanced data, restricting the model to OR@6 (and when needed, OR@3) faults is an appropriate approach [4].

First, the data was imported from the repository, and all unused signals were removed, as aforementioned. Thus, as there were 12 files for each fault at a given size, a total of 112 ten-second-long time-series files were used (108 being faulty, equally split between IR, B, and OR faults, and 4 normal time-series, one at each speed). All data was stored as a nested dictionary, containing data file names in the form: RPM_Fault_Size_LocationFrequency as outer keys. The outer values were also dictionaries, containing the location at which the data was collected from ("DE" or "FE") as keys, and NumPy arrays containing the corresponding time-series data as values. The complete data structure is illustrated as follows:

signals_dict = {1730_B_7_DE12: {"DE": np.array, "FE": np.array}, 1730_B_7_DE48: {"DE": np.array, "FE": np.array}… {…}

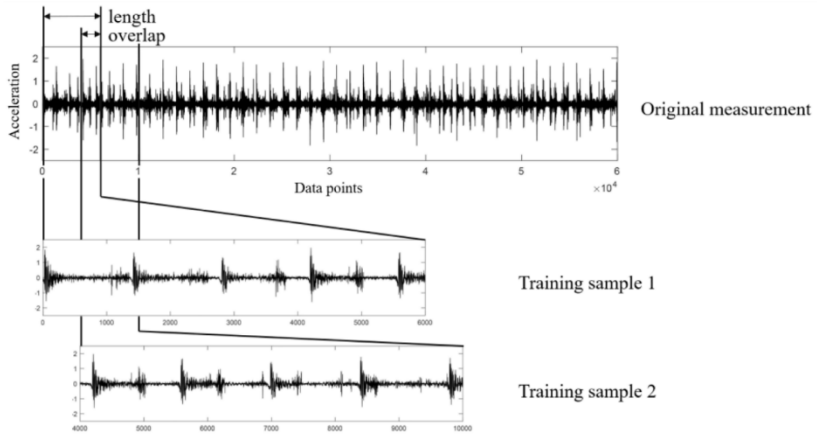

Figure 2: Illutsration of data segmentation into windows, from Hendriks et al [5].

Since some measurements were taken at a sampling rate of 48 kHz (namely the "Normal" measurements and data files ending with "48"), the "DE" and "FE" time series for these were downsampled by a factor of four to prevent certain data files from having an inflated influence on the classifier's training [6]. After the 48 kHz time series' were downsampled to 12 kHz, they were segmented into windows with some amount of overlap. *Figure* 2 illustrates this segmentation and overlap. The segmentation of these time series into multiple smaller samples is common practice, especially when working with limited test data [5]. The loading, processing, and overlapping segmentation of the time-series data was performed through a set of helper functions in prepare_data.py. Segmentation increases the amount of data the model has to work with Since the bearings are rotating at around 1800 RPM (or 30 RPS), the bearings in the CWRU dataset would have rotated about 300 times in each ten-second time series. As such, there is evident value in extracting more than one single data point from 300 rotations. Likewise, having some amount of overlap further increases the amount of data available without significantly impacting the quality of it. The optimal exact size of the windows, however, does not have a simple answer. Although details regarding the window size and overlap chosen are discussed in the results section, domain knowledge can be applied to set the bounds for potentially optimal window sizes, before using an iterative approach to determine the ideal range. Given that the bearings rotate about 30 times a second, with a sample rate of 12000 samples per second, an absolute lower bound of 400 samples in a window was set as smaller windows would fail to completely capture even a single rotation of the bearing. This absolute lower bound of 400 was doubled to 800 to account for overlap, and then increased to 1000 for simplicity (results show that this did not lose any

generality, as the optimal window size was well above 1000). A tentative upper bound of 12000 samples in a window was set (accounting to each window being 1 second long). Afterwards, both the Naïve Bayes classifier and Hidden Markov Model classifier were trained and tested with different window sizes between 1000 and 12000. Accuracies and other performance metrics were gathered for different overlap-window size combinations, and optimal ranges for these parameters were converged upon.

Naïve Bayes Classifier

Naïve Bayes is a probabilistic algorithm based on Bayes' theorem with the strong assumption of conditional independence among input features. This is a significant assumption that often hinders the efficacy of Naïve Bayes classifiers, given that very few features are truly independent in a real-world setting. Despite this simplifying assumption, Naïve Bayes classifiers are often effective, computationally efficient, and highly interpretable. The classifier computes the probability of each class given a set of input features and then selects the class with the highest posterior probability.

The Naïve Bayes classifier was implemented using scikit-learn. Both time-domain and frequency-domain (through a Fast Fourier Transform; FFT) features were extracted from the windowed and segmented vibration signals. This was done in extract_features.py, which extracted relevant time and frequency-domain features from each window, before bringing them into workable numeric arrays.

The following time-domain features were extracted:

- Mean amplitude.
- Standard deviation.
- Root mean square (RMS) of the waveform.
- Peak to peak value: the difference between the maximum and minimum amplitude.
- Crest factor: the ratio of the signal's peak amplitude to its RMS value.
- Kurtosis: the third standardized moment quantifying the asymmetry of the waveform.
- Skewness: the fourth standardized moment quantifying the peakedness of the waveform.

The following frequency-domain features were extracted after applying an FFT on the signals:

- Dominant frequency: the frequency component with the highest magnitude.
- Spectral centroid: the average frequency of a signal weighted by spectral magnitude, indicating where most of the energy is concentrated.
- Spectral bandwidth: the spread of frequencies around the spectral centroid.
- Peak FFT magnitude: the highest amplitude in the frequency spectrum.
- Total energy: the sum of squared magnitudes in the entire spectrum (or window).

These features are then normalized and stored in a matrix. Each row of this matrix corresponds to a single window of FE and DE signals, with the corresponding columns containing numeric values of each feature for that given window. It is important to note that the DE and FE channels of a single time series are always kept together, even after segmentation. This means that the model considers the data from the DE and FE readings at a given window as a single data point. From an architecture standpoint, this means that a single row (corresponding to one window) in the normalized master matrix contains time-domain (or frequency-domain) features of both DE

and FE readings of that window. For instance, (if time-domain features are selected) it would contain normalized values for the mean, standard deviations, RMS, and so on for the DE signal and contain normalized values for the mean, standard deviations, RMS, and so on for the FE signal, amounting to 14 columns in total. This serves to provide the classifier with complete information about a given window, as the Fan End signal at a given timeframe will likely be related to the concurrent Drive End signal in some way.

In order to avoid data leakage, signals from the 0.007” and 0.014” size faults are used in training, while signals from 0.021” size faults are reserved for testing. This ensures that the same faulty bearing does not appear in both training and testing sets, forcing the model to learn to classify near bearings’ signals, rather than simply memorize familiar, labelled bearings’ vibration signals. This prevents data leakage. Normal signals are randomly split in a 2:1 train:test split, to remain consistent with the split of the fault signals.

A 2:1 train-test split was deliberately selected for this project (as a consequence of an attempt to avoid data leakage), resulting in a somewhat more test-heavy dataset compared to common practice, which typically favors larger training sets (often around 80% training). This choice is justified primarily by the dataset size and the project's primary objective of evaluating the models' ability to generalize to unseen fault conditions. Due to the extensive amount of data available in the CWRU dataset and the fact that each fault size yields numerous segmented windows, even this relatively smaller proportion of training data provides ample instances for the classifiers to learn effectively.

Hidden Markov Model

A Hidden Markov Model (HMM) is a probabilistic model particularly effective at capturing sequential dependencies and temporal dynamics within time-series data. An HMM comprises hidden states and observable outputs, with transitions between states governed by probabilities. Observations depend probabilistically on these hidden states, allowing the model to represent complex temporal sequences through simpler state transitions and emission probabilities.

While this dataset might not be conventionally suited to an HMM, as there are no class transitions during the time series, it is still a tractable model for this problem. Although there are no class transitions (a bearing does not break during operation), there still are “hidden” state transitions. For example, a bearing with a specific type of fault would likely have a predictable change in its vibration signal as the damaged section aligns itself with the load. An HMM can model these state transitions and use them to predict the class of a bearing. Attempting to apply an HMM in a somewhat unconventional fashion also serves to highlight the potential versatility of HMMs, a matter that is rarely discussed in literature.

In this project, the HMM was employed to model the inherent temporal structure of bearing vibration signals, as periodic impacts and transient behaviors characteristic of bearing faults manifest clearly in such sequential models. The segmented time-series windows, which contained raw vibration data from both DE and FE sensors, served directly as input sequences to the HMM without additional feature extraction, preserving the temporal integrity of the signals.

Since this is a multi-class classification problem, separate HMMs were trained for each class (IR, B, OR, Normal). As was the case in the Naïve Bayes classifier, windows with fault sizes of 0.007” and 0.014” were used for training while windows with fault sizes of 0.021” were reserved

for testing to prevent leakage. Training each fault-specific HMM individually ensured that the model effectively learned the distinct temporal patterns associated with each fault condition. Once trained, classification of new test sequences was performed by evaluating the likelihood of a given vibration signal under each fault-specific HMM. The model assigned the fault classification corresponding to the HMM yielding the highest likelihood. Hyperparameter tuning, namely the number of hidden states, covariance type, and number of iterations the Expectation-Maximization algorithm will run, was performed manually. In addition, dimensionality reduction through principal component analysis (PCA) was considered. In both models, various performance quantifiers (ie, accuracy, recall, precision, f-score) were calculated. A confusion matrix and classification report with the aforementioned values were generated.

Results and Analysis

As stated, a Naïve Bayes model was selected as a classical, data-driven technique. In order to determine the optimal window size, overlap, and data type (time-domain or frequency-domain) for the Naïve Bayes classifier, a Naïve Bayes classifier was trained multiple times with different window sizes and overlaps. In accordance with the aforementioned methodology, window sizes of 1000 to 12000 were tested, incrementing by 500. Additionally, overlaps of 0.25, 0.5, and 0.75 were tested. While the accuracy of the Naïve Bayes classifier did not meaningfully change with overlap size, window sizes between 2000 and 3000 typically yielded marginally better accuracies than larger or smaller windows. *Figure 3* shows the accuracy of the classifier with a 0.5 (50%) overlap using time-domain features. It should be noted that accuracy peaks at a window size of 2000-2500, which is a consistent phenomenon regardless of overlap or random seed. *Figure 4* shows the accuracy of the classifier with the same overlap, but using frequency-domain features. As predicted, accuracy dropped steeply for smaller windows, likely due to potential data loss at small window sizes.

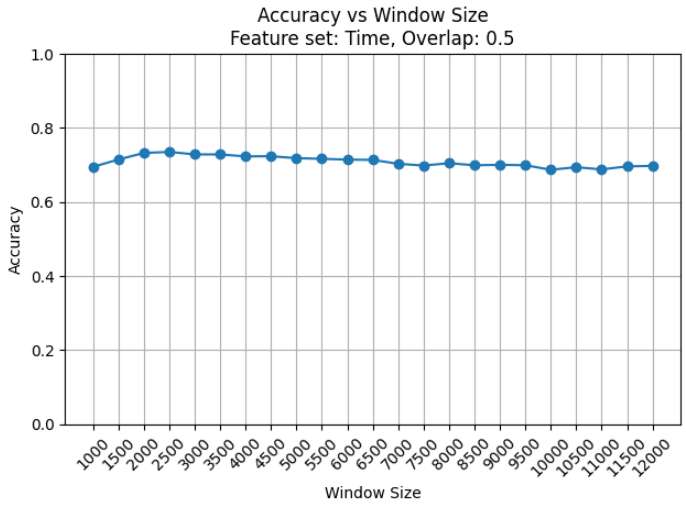

Figure 3: Accuracy vs window size graph using time-domain features

However, using frequency-domain features resulted in a greater tolerance towards smaller window sizes, while accuracy collapsed at very small window sizes when using time-domain features. Conversely, while accuracy dropped slightly when using larger than optimal window sizes with time-domain features, the accuracy drops at large window sizes when using frequency-domain features were significantly more pronounced. When comparing time-domain and frequency-domain features, classification using time-domain features consistently performed better than classification using frequency-domain features. Accuracies when using frequency-domain features peaked at around 65% when the window size was optimized. Contrarily, accuracies when using time-domain features peaked at around 73% with the optimal window size. It should be noted that the accuracy of the Naïve Bayes classifier could be “p-hacked” well into the high 80%/low 90% range by selectively omitting features due to the inherent assumption of feature independence.

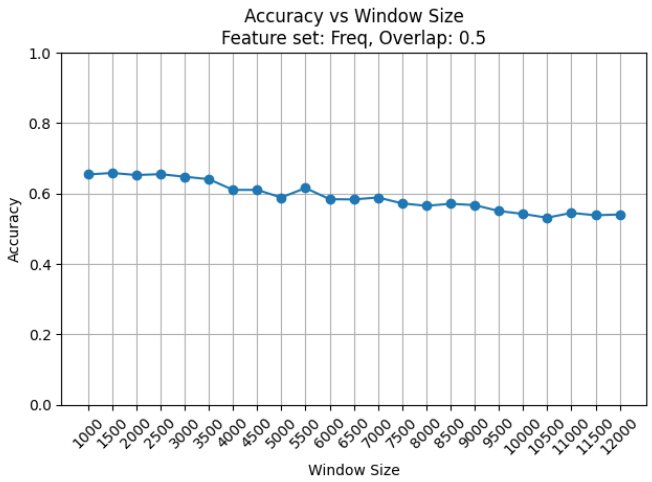

Figure 4: Accuracy vs window size graph using frequency-domain features

For example, the accuracy could be greatly increased by iteratively omitting features from the input vector and only including those that increase the accuracy. Additionally, priors (the only hyperparameter in a Naïve Bayes model, corresponding to the predicted class frequency) could be set to "guide" the model into classifying a predetermined portion of bearings into each class. However, both techniques were avoided, as they would fail to perform well with new data. Arbitrarily omitting features that might prove useful with new data, or forcing the classifier to preset class frequencies when we would not know them in a real-world application (although we did in this project) would likely have detrimental effects on broader model performance if faced with new data.

As an aside, it should be noted that using smaller windows and a greater overlap increased the training time. This is simply due to the impact (increase) these actions have on the dataset size. While previous work occasionally uses very large overlaps (such as 97%), this was deemed unnecessary for the Naïve Bayes classifier, as it had no noticeable effect on the performance of the classifier [4]. Given that the Naïve Bayes classifier is only one part of this project, all future comparative discussion will only refer to the Naïve Bayes classifier using time-domain features with optimal window sizes, as they have presented the best performance.

In contrast to the Naïve Bayes classifier, a Hidden Markov Model was selected as a modern machine learning technique. To maintain some degree of simplicity, segmented windows (in the same structure as the ones used for the Naïve Bayes classifier) were directly fed into the HMM, rather than any transformed features. Further work could potentially explore the performance of an HMM on this data after it has been transformed in different ways, such as a wavelet transform or short-time Fourier transform, noting that the latter has yielded strong results when using classical techniques like a Multi-layer Perceptron classifier and a Support Vector Machine [5]. A similar iterative method was used to determine the ideal window size and overlap for the HMM. Once again, it was observed that while the HMM classifier was not meaningfully sensitive to overlap size, window sizes of 2000-4000 yielded the best accuracy. This is close to its analogous Naïve Bayes optimal window size range, which is likely due to the nature of the data being similar, given that they are both non-transformed time series.

The HMM's hyperparameters were manually tuned. The model was run with different permutations of hyperparameters, and the values of hyperparameters that maximized the accuracy of the classifier were chosen. The following hyperparameters were tuned:

- Number of hidden states. Domain knowledge suggested that the optimal number of hidden states was around 3, due to the "rise, peak, fall" pattern of the bearings' signals. As expected, 3 hidden states provided the greatest accuracy, with a greater (or lesser) number significantly reducing accuracy.

- Covariance type. Covariance type determines how each state's covariance matrix is modeled. "full" was an obvious choice, as it permitted each state to have its own full covariance matrix (with cross-terms). Other options, like "diag" and "tied" would limit the covariances, or in the extreme case of "spherical" would assume all features are uncorrelated. As expected, "full yielded" the best performance by a significant margin.
- Number of iterations. This hyperparameter was easy to tune. Different values, ranging from 30 to 200, were tested, and it was determined that the model had converged by 100 iterations (additional iterations would not increase the accuracy). Hence, 100 iterations are used in the EM algorithm.
- PCA components. The option to apply a form of dimensionality reduction called Principal Component Analysis to the training data before fitting the HMM was presented. However, it was determined that reducing the 2-dimensional DE-FE signal pair into a 1-dimensional subspace significantly reduced accuracy. This is due to the significant differences between the DE and FE waveforms, as the faulty bearing is only placed on one of the two ends (so the waveform of the other end would be more standard). Dimensionality reduction would have resulted in the loss of this information, so it was avoided.

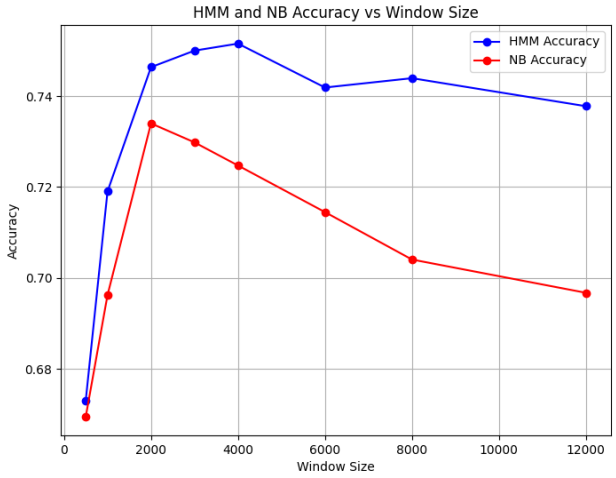

Figure 5: HMM and Naïve Bayes accuracy at different window sizes

Results indicate that the HMM classifier was modestly, but consistently, more accurate than the Naïve Bayes classifier. *Figure 5* shows the performance of the HMM in comparison to the Naïve Bayes classifier at different window sizes and aforementioned hyperparameters (with an overlap of 0.5). Two important results can be noted from *Figure 5*. Firstly, although the margin was small, the HMM consistently outperformed the Naïve Bayes classifier at all tested window sizes. Secondly, the HMM maintained its accuracy over a larger range of window sizes. While the Naïve Bayes classifier's accuracy quickly dropped as window sizes grew past 2000, the HMM remained almost as accurate as its peak at the maximum test window size of 12000. This result is even more impressive in the context of past work on the CWRU dataset. Despite being applied in a somewhat unorthodox context, due to the lack of obvious state shifts in the signals, the HMM outperformed various CNNs in the absence of data leakage. These results highlight the flexibility of HMMs and potentially contribute to their versatility in various applications.

Accuracy was not the only performance metric evaluated. Precision, Recall, and F-Scores were also evaluated for each class in both models. They are defined as such:

$$Accuracy = \frac{True\ Positive + True\ Negative}{Total} \qquad Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \qquad F - Score = 2\frac{Precision \times Recall}{Precision + Recall}$$

In addition, confusion matrices were also generated to better analyze the type of fault the models succeeded (and failed) in catching. *Figures 6* and *7* present classification reports containing all of these parameters for the Naïve Bayes classifier and HMM, respectively. Multiple conclusions can be drawn from this. Firstly, both models were extremely successful in classifying "Normal" (undamaged) bearings. This is partially due to the inevitable data leakage that occurs from randomly splitting the "Normal" labelled data into training and testing sets (as all Normal signals come from one single physical bearing), and likely partially due to the relative ease in recognizing a relatively smooth and consistent waveform. It can also be noticed that the Naïve Bayes classifier primarily failed to recognize OR faults, while the HMM's class-wise performance (as quantified by the f1-score, the harmonic mean of precision and recall) was more evenly distributed between the three faulty classes. One potential reason for this is that some OR fault files were OR@3 instead of OR@6 (as the latter was not present for a given size/speed). This highlights the difficulty naïve models have in "transferring" knowledge from one well-learned class to another, similar, new class.

```
=== Naive Bayes Classification Results ===

Classification Report:
              precision    recall  f1-score   support

           B       0.76      0.82      0.79       916
          IR       0.76      0.72      0.74       914
      Normal       1.00      1.00      1.00       104
          OR       0.63      0.62      0.62       916

    accuracy                           0.73      2850
   macro avg       0.79      0.79      0.79      2850
weighted avg       0.73      0.73      0.73      2850

Accuracy: 0.7298
Confusion Matrix:
 [[750  90   0  76]
 [  0 660   0 254]
 [  0   0 104   0]
 [237 113   0 566]]
NB Accuracy for window size 3000: 0.7298
```

Figure 6: NB Classification Results

```
=== HMM Classification Results ===

Classification Report:
              precision    recall  f1-score   support

           B       0.89      0.62      0.73       916
          IR       0.67      0.87      0.76       914
      Normal       1.00      1.00      1.00        86
          OR       0.73      0.73      0.73       916

    accuracy                           0.75      2832
   macro avg       0.82      0.81      0.81      2832
weighted avg       0.77      0.75      0.75      2832

Accuracy: 0.7500
Confusion Matrix:
 [[569 216   0 131]
 [  1 799   0 114]
 [  0   0  86   0]
 [ 70 176   0 670]]
HMM Accuracy for window size 3000: 0.7500
```

Figure 7: HMM Classification Results

Finally, model training time must be mentioned. The inherent assumption of feature independence in the Naïve Bayes classifier makes it significantly faster than the HMM classifier. Training time was measured through the tqdm library. All training was done on an AMD Ryzen 5 3600 6-core processor. A GPU was not utilized for training. The Naïve Bayes classifier took about 5 seconds to extract features (which can be done separately) and 0-2 seconds to train. In contrast, an HMM with 3 hidden states took between 10:11 and 18:40 minutes to train. Increasing the number of hidden states would also further increase the training time, at the order of $O(N^2)$ [8]. That being said, compute optimization (such as using a library amicable to multi-core CPU compute, or GPU compute) is likely to reduce training time significantly.

Conclusion

To conclude, comparing a multiclass Naïve Bayes classifier and Hidden Markov Model resulted in various novel findings. A Naïve Bayes classifier was trained using time-domain features, while a Hidden Markov Model was trained using the raw segmented time-series signals. Data leakage was minimized by segregating separate bearings by train and test set, preventing a single bearing's signal from appearing in both. However, an inevitable limitation is that there was some amount of data leakage with the healthy signals, as all came from one single physical bearing.

Despite being relatively noncomplex, with minimal pre-processing done to the data fed into the HMM, it proved to be consistently more accurate and able to process larger window sizes with

minimal impact on performance. This came at the expense of a significantly longer training time, and some difficulty with regard to ease of use.

The outcomes of this study enable a promising area of future work. Although many classifiers struggle to operate on the CWRU dataset, it is possible that a further optimized HMM (such as through FFTs or wavelet transforms) can perform even better, while minimizing the occurrence of data leakage. Additionally, it highlights the versatility of Hidden Markov Models. While they are typically utilized only in contexts where clear, explicit state changes (such as the vibration signal of a bearing that breaks *during* the period of recording), it has been demonstrated that it is possible to apply it to problems where state changes are less overt.

Works Cited

[1] CWRU Bearing Data Center, “Case Western Reserve University Bearing Data Center,” [Online]. Available: https://engineering.case.edu/bearingdatacenter. (Accessed: 30-Mar-2025).

[2] F. Jia, Y. Lei, J. Lin, X. Zhou, N. Lu, Deep neural networks: A promising tool for fault characteristic mining and intelligent diagnosis of rotating machinery with massive data, Mech. Syst. Sig. Process. 72–73 (May 2016) 303–315, https://doi.org/10.1016/j.ymssp.2015.10.025.

[3] W. Zhang, G. Peng, C. Li, Y. Chen, Z. Zhang, A new deep learning model for fault diagnosis with good anti-noise and domain adaptation ability on raw vibration signals, Sensors 17 (2) (2017) 425, https://doi.org/10.3390/s17020425.

[4] Z. Zhu, G. Peng, Y. Chen, H. Gao, A convolutional neural network based on a capsule network with strong generalization for bearing fault diagnosis, Neurocomputing 323 (2019) 62–75, https://doi.org/10.1016/j.neucom.2018.09.050.

[5] J. Hendriks, P. Dumond, and D. A. Knox, "Towards better benchmarking using the CWRU bearing fault dataset," *Mech. Syst. Signal Process.*, vol. 169, p. 108732, 2022, https://doi.org/10.1016/j.ymssp.2021.108732.

[6] H. Abburi et al., "A Closer Look at Bearing Fault Classification Approaches," arXiv preprint arXiv:2309.17001, Sep. 2023, https://doi.org/10.48550/arXiv.2309.17001.

[7] S. Rigas, "CWRU_Bearing_NumPy," GitHub repository, [Online]. Available: https://github.com/srigas/CWRU_Bearing_NumPy/tree/main. (Accessed: March 30, 2025).

[8] E. P. Xing. (2020). "Lecture 6: Case Studies: HMM and CRF" [Online]. Available: https://www.cs.cmu.edu/~epxing/Class/10708-20/scribe/lec4_scribe.pdf

https://jonathan-hui.medium.com/machine-learning-hidden-markov-model-hmm-31660d217a61

https://web.stanford.edu/~jurafsky/slp3/A.pdf

https://hmmlearn.readthedocs.io/en/latest/tutorial.html

https://compneuro.neuromatch.io/tutorials/W3D2_HiddenDynamics/student/W3D2_Tutorial2.html

https://scikit-learn.org/stable/modules/naive_bayes.html

https://www.kaggle.com/code/prashant111/naive-bayes-classifier-in-python

https://www.statology.org/sklearn-classification-report/

GitHub Copilot was used while coding, primarily to produce docstrings and fix bugs.

ChatGPT used for the following purposes:

- Brainstorm frequency-domain features for the Naïve Bayes classifer.
- Software architecture, to organize functions and modules.
- Program log-likelihood maximization for the multi-class HMM.
- Print/present data in a clean way (ie. with "\n=== HMM Classification Results ===").
- Broad “sanity-checking” code. Ie. “I built this HMM classifier, does it make sense?”.

Appendix/Code

The following files are submitted along with this project. They can also be found in the following GitHub repository: https://github.com/imported-canuck/MECH-501/tree/main/Project. Simply cloning the repository and running run.py on the terminal should work.

run.py (this is the “single file” the grader can run by writing **python run.py**)

main.py (to retrieve custom results, ie. modify the number of hidden states)

HMMClassifier.py

NaiveBayesClassifer.py

extract_features.py

prepare_data.py

BearingData.py

Directory named “CWRU Bearing Dataset” containing the raw data files.

Note: All files and directories must be at the highest level with respect to run.py to run properly. The user must also have a “Downloads” folder.

The graphs in Figure 3 and 4 were generated via a different script. This script was an intermediate script (not used in the final submission). However, to regenerate the graphs (along with others), the grader can run main2.py directly. This run takes about 15 minutes.

For this to function, the user needs to have the following files in the highest level with respect to main2.py:

NBClassifier.py

ExtractFeatures.py

Preprocess.py

LoadSignals.py

BearingData.py

Directory named "CWRU Bearing Dataset" containing the raw data files.

Note: I do not suggest re-running run.py. The whole module takes about 2 hours to run. In lieu of that, the output is attached below. It is quite long. Attempting to rerun run.py should yield identical results, as a specific random seed was declared.