An Exploration of the Effect of Feature Selection on Machine Learning

Brendan Hawk

2022-12-13

The Data

The data prescribed for this analysis is a portion of the 2018 Behavioral Risk Factor Surveillance System (BRFSS) Survey Data prepared by the United States Department of Health and Human Services Centers for Disease Control[1]. Per the dataset overview:

> *BRFSS's objective is to collect uniform state-specific data on health risk behaviors, chronic diseases and conditions, access to health care, and use of preventive health services related to the leading causes of death and disability in the United States.*

The class to be predicted in this experimentation is the presence or absence of a medical diagnosis of arthritis. For this analysis, there are 11933 observations of 107 attributes and 1 class attribute in the raw dataset. The class to be predicted (`havarth3`) has two levels, and is moderately imbalanced with 33.747% positive cases.

Preprocessing and Imputation

The following preprocessing steps were taken, beginning with the raw dataset:

- The class values were recoded from 1 or 2 to "Positive" and "Negative" respectively, using appropriately ordered factor levels so that "Positive" would be the positive predictive case for all algorithms tested.
- Numeric columns were read as such and values of `?` were implicitly converted to missing, all others were read as strings.
- Several columns had minor recoding done by hand per information found in the codebook:
  - `children` had values 88 ("None") recoded to 0, and 99 ("Refused") recoded as missing.
  - `drocdy3.` had values of 900 ("Don't know/Not Sure Or Refused/Missing") recoded as missing.
  - `x.drnkwek` had values of 99900 ("Don't know/Not Sure Or Refused/Missing") recoded as missing.
- `drocdy3.`, `wtkg3`, `x.bmi5`, `html4`, `x.drnkwek` were all divided by 100, as these values have implied decimals according to the codebook.
- All non-numeric attribute values of `?` were recoded as missing, and categorical attributes were then turned into categorical factors.
- The following attributes were dropped because they are part of the originating study design, and are not predictors of the class attribute. In an analysis more specific to the data prescribed here, these would of course be critical pieces to creating statistical results that best represented the surveyed population. However, this analysis is focused on experimenting with attribute selection methods and model training so I believe they are safe to remove at this stage.

[1] Centers for Disease Control and Prevention (CDC). Behavioral Risk Factor Surveillance System Survey Data. Atlanta, Georgia: U.S. Department of Health and Human Services, Centers for Disease Control and Prevention, 2018.

  - `x.psu`
  - `x.llcpwt`
  - `x.llcpwt2`
  - `x.wt2rake`
  - `x.ststr`
  - `x.strwt`
  - `x.rawrake`
- Additionally, the `weight2` and `height3` attributes were dropped. It is evident from the codebook that these are duplicated in other measures in a way that would make them 100% correlated, and any attribute selection method would (or at least *should*) remove them. I could have left this for them to drop, but I chose to remove them myself here as part of the preprocessing.

At this point, a file was written (`BrendanHawk-project-initial.csv`). The dataset was then split roughly 66%/33% into training and test partitions. There was still a matter of imputing missing values, however it is critical that this happens after the training/test split. If the imputation methods take the test data into account, it begins to blur the lines between the partitions, and taints the purity of the test set as a valid measure for evaluating algorithms later.

Imputing the numeric attributes in the training data was done with random values from a normal distribution using the attributes' means and standard deviations. These parameters were then applied to create random values from the same distribution to impute missing values in the test set. In this way we preserve the idea that the training data informs our expectations of the test data but not vice versa. The numeric attributes were then also normalized.

Similarly, imputing the categorical values was done by randomly sampling the distinct values of such attributes from the training data with their prevalence used to form sample probabilities. The same probabilities, derived from the training data, were then used to randomly impute values in the test set as well.

As a last measure of data quality, the class distributions of the training and test datasets were checked against the original dataset to make sure that they remained similar through the random split.

Finally, both of these sets now having been fully preprocessed and imputed were written to files: `BrendanHawk-training.csv` and `BrendanHawk-test.csv` respectively.

Attribute Selection Approaches

All the attribute selection approaches applied here fall generally into the category of *filter* based selection algorithms. These are a class of methods that uses generalized descriptive statistics in some way to evaluate the expected usefulness each attribute might have in accomplishing the given classification or regression task. Compared with their counterparts, *wrapper* methods, filter-based feature selection is considerably less computationally expensive and quicker to calculate. Wrapper methods also rely on a given machine learning algorithm internally to do evaluation, which can bias the results towards that algorithm specifically, e.g. if you use a wrapper method based on the Random Forest algorithm, the results may work will for other RF models but not for a neural network or KNN model. Normally, filter methods select some number of attributes using a percentage, a cutoff, or some other heuristic. I, however, wanted to fully explore them and what they could provide as "best" solutions. To that end, I developed a testing process to consider every possible cutoff to find the "best of the best" for each selection method.

There is an incredibly diverse and seemingly unending array of methodologies for selecting and evaluating a feature selection method. In an effort to keep this report as consistent as possible, a kind of "benchmarking" approach was used. This ensures that each selection method was given exactly the same treatment and their results are completely reproducible. The goal of the benchmark was to evaluate as many options as possible, and select the one that has the fewest attributes that performed the best with a small tolerance to avoid over-selecting. This benchmark was applied to all except the last method (see "Correlation-based Feature Selection" below), as the implementation of that method leverages a built-in "best first" selector that closely

mimics (but is not exactly equal to) the elbow method applied to the rest of the evaluations. As a point of fact, this benchmarking process converts the filter based methods to something closer to a wrapper method for feature selection.

All of the implementations of these methods were leveraged from an `R` package called `FSelector`[2]. This is an excellent package with several feature selection methods as well as wrappers to help choose the number of features given the measures calculated for each attribute.

To begin the benchmark process, each attribute is given some form of score or value using the measures described below. Then, a selection of the k best of these were selected iteratively from 1 attribute to all attributes. For each number of attributes, a basic (untuned) classification tree was built. The AUROC score was used as a final scoring measure of this tree in lieu of the accuracy because of the class imbalance in the originating dataset. These final AUROC values were plotted against the number of features; see figures 1 through 5 in Appendix A: Figures. These plots were manually evaluated (and annotated accordingly) using the elbow method, to determine the fewest attributes possible without meaningfully sacrificing performance.

The following five attribute selection methods were tested:

Information Gain Ratio Information Gain Ratio is a metric that comes from tree-based partitioning algorithms. It is a ratio calculated from two values: the expected gain of information partitioning the dataset by a given attribute and a measure of the intrinsic information represented by that attribute. As a measure used for feature selection, this ratio is calculated once for each attribute in the dataset, and the highest k values were tested at each iteration of the benchmarking process.

18 attributes were selected by this method: **diffwalk, diffdres, chccopd1, x.age65yr, x.rfhlth, x.age80, diffalon, x.age.g, x.hcvu651, employ1, chckdny1, pneuvac4, cvdcrhd4, x.phys14d, x.michd, x.exteth3, deaf, genhlth**

Chi Squared Pearson's Chi-Squared test evaluates the likelihood of combinations of attribute values and class values happening by chance. It is calculated from the observed counts of each set of discrete variables and the counts of each distinct combination of said variables. The method implemented here is an extension of this known as Cramer's V and has the following formula:

$$V = \sqrt{\frac{X^2/n}{min(r-1,k-1)}}$$

This becomes a measure of the interrelatedness of two discrete variables. As with the Gain Ratio, the highest k values were tested at each iteration of the benchmarking process.

6 attributes were selected by this method: **x.ageg5yr, x.age80, x.age.g, employ1, diffwalk, genhlth**

One-R One-R is another approach that comes from partitioning algorithms. This algorithm makes a simple set of rules based on the distinct values of each attribute separately, and an error rate is calculated using just that one set of rules to classify the entire training dataset. In contrast with the previous measures, the attributes corresponding to the *lowest* k of these error rate values was tested at each iteration of the benchmarking process.

12 attributes were selected by this method: **diffwalk, employ1, physhlth, x.phys14d, genhlth, x.rfhlth, chccopd1, x.ageg5yr, x.age65yr, diffalon, rmvteth4, x.age.g**

RReliefF RreliefF[3] is an extension of the Relief algorithm: a process of evaluating the "heuristic merit" of attributes in a given dataset. In short, this is done iteratively by selecting random observations from the

[2]Romanski P, Kotthoff L, Schratz P (2021). *FSelector: Selecting Attributes*. R package version 0.33, https://CRAN.R-project.org/package=FSelector.

[3]Robnik-Sikonja, Marko & Kononenko, Igor. (2000). An adaptation of Relief for attribute estimation in regression. ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning.

training data and one (or more) of the most similar observations that have the same class and a similar match (or matches) that have dissimilar classes. For each of these "hits" and "misses" a weight is updated for each attribute.

Again, the highest k of these values was tested at each iteration of the benchmarking process.

21 attributes were selected by this method: **diffwalk, persdoc2, renthom1, sex1, x.race, flushot6, genhlth, x.race.g1, x.racegr3, x.imprace, qstver, x.phys14d, diabete3, x.ment14d, rmvteth4, x.rfbmi5, hlthpln1, exerany2, x.totinda, x.michd, x.age.g**

Correlation-based Feature Selection This is another "heuristic merit" evaluation that is based on the correlation of attributes to the class as well as their intercorrelation. From the originating paper[4]:

> *The numerator ... can be thought of as providing an indication of how predictive of the class a set of features are; the denominator of how much redundancy there is among the features.*

The implementation leveraged here is built with a "best" search internally, and so this measure was not run through the same benchmark as all the rest. This search, however, loosely follows the same principle of choosing the smallest subset of features without sacrificing a meaningful amount of data using a "forward search" methodology.

12 attributes were selected by this method: **employ1, pneuvac4, diffwalk, diffdres, diabete3, persdoc2, chccopd1, x.age80, x.age65yr, x.rfhlth, x.phys14d, x.exteth3**

Classfication Algorithms

I chose to use the `caret`[5] package in `R` to further ensure that there is a great deal of consistency, reproducibility, and opportunity for direct comparison between the 25 models created. This package neatly wraps the training, tuning, and testing processes of model building and provides a number of ways to control how each stage is done. In this analysis, the tuning controls were identical between all models built: 10-fold Cross validation was performed on the training set using the same folds for each model, AUROC was used as the test measure for each cross validation test, the random seed was reset before each model was trained, and no individual model was tuned by hand in any way. There is, by default, some amount of default tuning done "under the hood" for each algorithm applied. I didn't feel the need to specifically limit this nor did I think it was prudent to specifically fine-tune any individual model. In practice, after generally testing each algorithm and selecting one or two that work best for my task I would go back and spend more time tuning the final models for comparison. In this way, this analysis really specifically highlights the first two stages of model development (feature selection, general model experimentation) more than the last stages of producing a highly tuned model and applying it to future unobserved data in the real world.

For each combination of algorithm and features, a model was trained in this way on the training dataset and final performance evaluations were measured against the test dataset. Several metrics were collected: Accuracy, TPR, FPR, Precision, Recall (equivalent to TPR), F1, F2, AUROC, and the Mathew's Correlation Coefficient. All model evaluation results are presented in Appendix B: Model Results.

[4]Hall, Mark A. 1999 *Correlation-based Feature Selection for Machine Learning*, The University of Waikato, Hamilton, New Zealand

[5]Kuhn M (2022). *caret: Classification and Regression Training.* R package version 6.0-93, https://CRAN.R-project.org/package=caret.

The following algorithms were evaluated:

J48 While this analysis is being performed in `R`, I thought it would be interesting to be able to compare results to one of the most common tree algorithms used in Weka as well. Luckily, there are integrations in the `RWeka` package[6] that allow this. J48 is a Java implementation of the C4.5 tree algorithm, which in itself is an extension of the ID3 tree algorithm. In short, this algorithm recursively partitions the tree on a single attribute, chosen at each iteration by which of the remaining attributes has the highest Information Gain Ratio value.

C5.0 Internally, there are a number of notable updates when comparing C4.5 (implemented as J48) and C5.0. Most notable for this analysis, however, is that C5.0 leverages boosting: multiple trees are built sequentially, and sample weights are increased when they are misclassified so as to better inform the next tree built and (hopefully) increase the overall efficacy of the final learned model. Higher performing trees built inside this approach are also given higher weight in the final model. This is the only algorithm that required intervention in the tuning stages, because there is a built-in option to "winnow" the features used for training these models. As that would taint or defeat the purpose of experimenting with feature selection methods I chose to prevent this.

Naive Bayes Naive Bayes classifiers are amongst the simplest, in theory. They use the aggregation of Bayes' Theorem outputs on the given attributes to predict the probability of each class, and the higher value becomes the prediction.

K Nearest Neighbors KNN is a distance-based algorithm that tries to predict the class of a given observation by looking at its k most similar neighbors - whichever class is most represented in that sample becomes the class vote for that observation.

Linear SVM In short, Support Vector Machines attempt to perceive the given dataset in higher dimensions so as to be able to discover an optimal hyperplane to linearly separate the class groups. Particularly of note is that for categorical data (which is the vast majority of the given dataset), the data must be encoded with something like one-hot or dummy encoding. This means that the required processing of our datasets is *significantly* expanded, however most modern SVM solvers can accept an astronomical number of dimensions with ease.

Overall Findings

Overall, it's clear that the less holistic performance measures are not always great at distinguishing "good" models. This isn't news, but this experimentation certainly underscored it heavily. Accuracy alone is particularly unhelpful, as seen in Fig. 5. Precision is not much better at distinguishing as seen in Fig. 8. As I will discuss later, it was critical to use more complex measures, such as the AUROC and MCC measures, to distinguish between the performance of each model. It was also underscored how much importance there is in using *multiple* measures; neither the AUROC or MCC alone would have been sufficient in isolation to choose a single model.

While classification trees are known to be highly efficacious, I will admit that I am surprised at how consistently they performed regardless of attribute selection. Particularly in the C5.0 models, the differences are almost too difficult to pull apart by eye alone in the figures of performance measurements. That may give rise to future work looking at the common attributes selected amongst all attribute selection methods and trying to use only those to build a performant model that could be easily understood such as a standard

[6] Hornik K, Buchta C, Zeileis A (2009). "Open-Source Machine Learning: R Meets Weka." *Computational Statistics*, *24*(2), 225-232. doi:10.1007/s00180-008-0119-7 https://doi.org/10.1007/s00180-008-0119-7.

tree. Even if there is some loss in accuracy, it could become easy enough to be a first-pass heuristic for a medical professional to order more exact tests to diagnose a patient's potential arthritis.

With any mix of attribute selection experiments, inspecting commonly chosen attributes can be a source of insight for further analysis. The following table presents the 5 most commonly chosen attributes and how many different selection methods chose them:

Table 1: Top 5 Chosen Attributes

Attribute	Num. Times Selected
diffwalk	5
employ1	4
genhlth	4
x.age.g	4
x.phys14d	4

It is no surprise that `diffwalk` is selected for in all 5 cases. Per the codebook, this attribute is the response to the question:

> *Do you have serious difficulty walking or climbing stairs?*

This is an obvious and common symptom to arthritis, especially arthritis of the hips, knees, and back. According to the National Arthritis Data Workgroup[7] "More than 22% of American adults (over 52.5 million people) have arthritis or another rheumatic condition diagnosed by a doctor." Similarly, `x.age.g` (a binned age attribute), `x.phys14d` (how many of the previous 14 days was physical health "not good"), and `genhlth` (a judgement of ones overall physical health) would also be somewhat obvious predictors of someone suffering the symptoms of arthritis. On the surface, the common inclusion of `employ1` may be less obvious. Digging deeper, however, we find that some of the categories might be obvious predictors of the chance of arthritis, such as `Retired` and `Unable to work`.

While this is a sound enough way to pick some of the "most important" attributes, it's possible, and even likely, that a researcher with domain knowledge might pick other important attributes that they thought were most likely correlated with the presence of arthritis. In practice, machine learning can only come so close to truth without expert knowledge informing it.

Final Model Selection

When evaluating the models, as well as when building them, I was aware of the class imbalance in the original dataset. As such, I wanted to make sure that I wasn't relying on an uninformative metric through any stage of attribute or model selection. To this end, every step of the process was configured to use the AUROC values as determining or evaluating scores for all coded implementations. When evaluating final performance metrics, I also took the F2 and MCC scores into account. Particularly, the MCC scores showed some very poor performance in models that were otherwise highly scored by other measures, and made C5.0 appear to be the best performing algorithm overall.

It was exceedingly difficult to pick a single best model given the results of this analysis. Though it was clear enough that the C5.0 algorithm performed the best all around, it confusingly did so with seeming disregard for which attribute selection results it was given. In the corresponding figures and tables, however, it is (barely) clear that the Gain Ratio, OneR, and RreliefF methods outperformed the other two by a small margin in the F2 and AUROC measures. With that in mind, I wanted to verify as soundly as possible that there was a reason to choose one in particular.

[7] Eustice, Carol. "Arthritis Prevalence and Statistics." Verywell Health, Verywell Health, 4 Feb. 2022, https://www.verywellhealth.com/arthritis-prevalence-and-statistics-189356.

Table 2 below presents a one-way ANOVA that shows the selector method is significantly correlated with a difference in mean efficacy during cross validation. This analysis was computed from the AUROC values calculated during the cross validation stage of model building. Recall that the model building implementation was designed to ensure that the 10 folds were the same across all models built, so we're able to do direct comparisons like this.

Table 3 similarly shows pairwise comparisons of the means, using the Bonferonni adjustment for the p values. For both tables below, the p values aren't actually zero of course, but are extremely close to zero and are displayed as such to rounding.

Table 2: AOV: Cross-Validated AUROC Values vs Feature Selection Method

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Feature.Selector	2	2.49e-04	1.25e-04	2328	0
Residuals	24	1.29e-06	5.40e-08		

Table 3: Comparison of Means with Bonferonni Adjustment

	diff	p adj
OneR-Gain Ratio	-0.00427	0
RreliefF-Gain Ratio	0.00314	0
RreliefF-OneR	0.00742	0

We can see that there is statistically significant evidence to support a conclusion that according to the analysis of the AUROC values, the OneR attribute selection method in conjunction with the C5.0 model performed the best of all 25 models.

12 attributes were selected by the OneR method and used for this final C5.0 model: **diffwalk, employ1, physhlth, x.phys14d, genhlth, x.rfhlth, chccopd1, x.ageg5yr, x.age65yr, diffalon, rmvteth4, x.age.g**

The final measures of performance for this model were as follows:

Table 4: Confusion Matrix: C5.0 Algorithm, OneR Feature Selection

	Positive	Negative
Positive	668	336
Negative	701	2352

Table 5: Results: C5.0 Algorithm, OneR Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.744	0.665	0.230	0.488	0.665	0.563	0.620	0.795	0.398
Negative	0.744	0.770	0.335	0.875	0.770	0.819	0.819	0.795	0.398
Wtd. Avg.	0.744	0.735	0.299	0.744	0.735	0.733	0.752	0.795	0.398

The reduced training and test datasets used for this model were also written to their own files, `BrendanHawk-best-train.csv` and `BrendanHawk-best-test.csv` respectively.

Summary and Final Conclusions

Overall, this analysis demonstrated the importance of each step taken when building machine learning algorithms. Attribute selection is a complex and expansive task. The measures used in evaluating importance are critical, and so is gleaning what the results actually mean using expert knowledge and references to the data sources and their design. For example: without the codebook I would not have been able to figure out why employment status had been selected by 4 out of 5 attribute selection methods. An analyst must always try to have as much of the contextual information as possible, "the qualitative behind the quantitative", or a vast majority of the applicable results may be lost. At the very least, opportunities for future work and deeper insights may be out of reach.

Equally important, of course, is the selection of the algorithm used to perform the analysis. It's clear that tree-based algorithms' reputation for being powerful classifiers is well-earned. I am not entirely surprised that the Naive Bayes and KNN approaches did not perform as well.

Which brings me to another point worth repeating: the need for multiple and complex performance evaluation measures. It is simple to show that accuracy is only a very rough guide, and does not paint the whole picture of an algorithm's performance. This is especially true of imbalanced datasets. This dataset is only moderately imbalanced, about 33% Positive cases, but there are a great many times in practice when a target dataset is much more highly skewed. Prediction on medical data, for example, can often be limited to prevalences of less than 1%. Measures such as F-scores, AUROC, PRC, and MCC are critical to evaluating the overall success.

One part missing from this exploration that interests me is the idea of evolving a model over time. This data is admittedly several years old, and more observations probably are not available as the survey may have been redesigned since then. If not, it would be fascinating to see how a model can be continuously informed by further observations brought in after initial training. There is a large volume of experimentation and research trying to design methods for updating models without having to re-train them from scratch, and in which cases such an approach would be efficient or useful.

Appendix A: Figures

Fig. 1: Feature Evaluation Results attrs.gain.ratio.tests

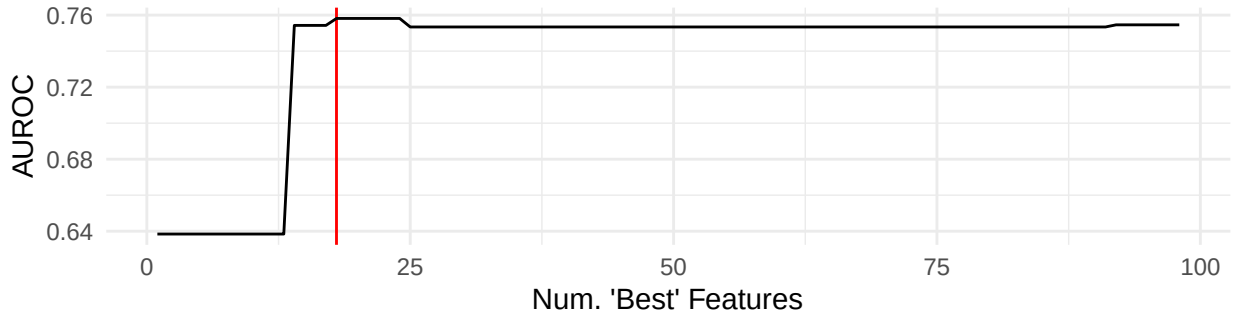


Fig. 2: Feature Evaluation Results attrs.chi.squared.tests

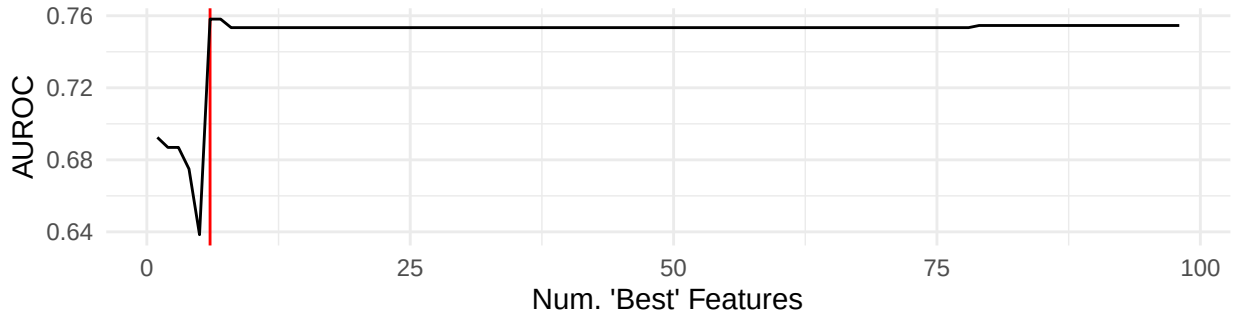


Fig. 3: Feature Evaluation Results attrs.oneR.tests

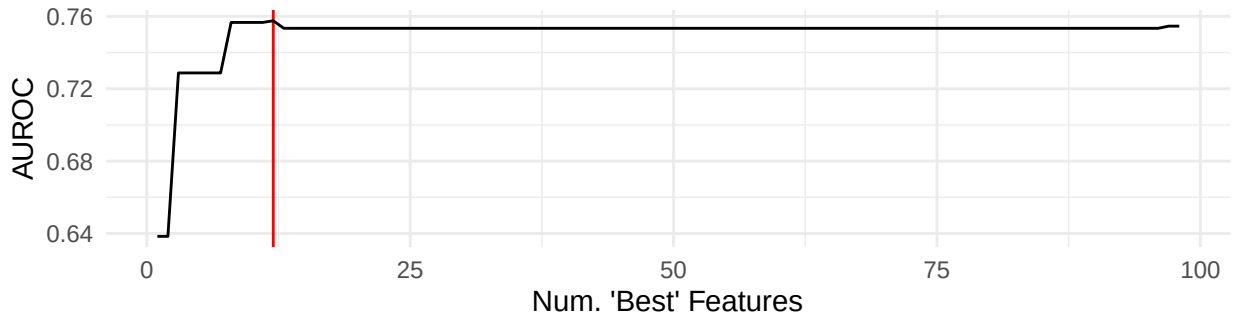


Fig. 4: Feature Evaluation Results attrs.relief.tests

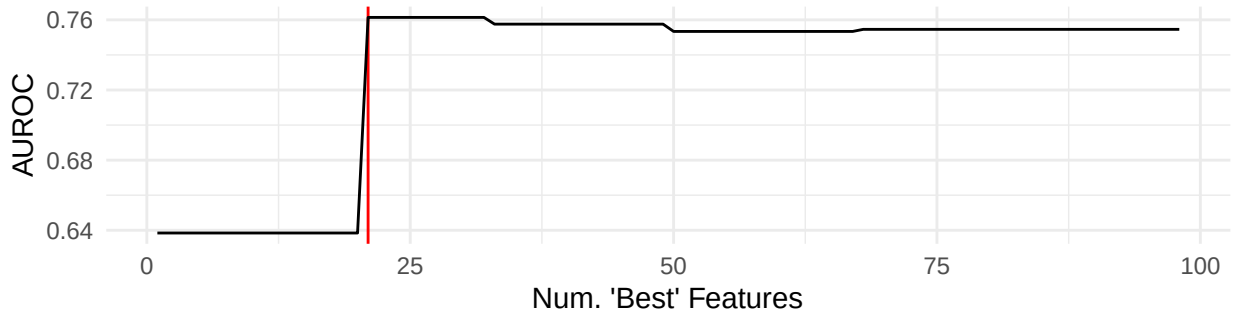


Fig. 5: Comparison of Model Performance: Accuracy

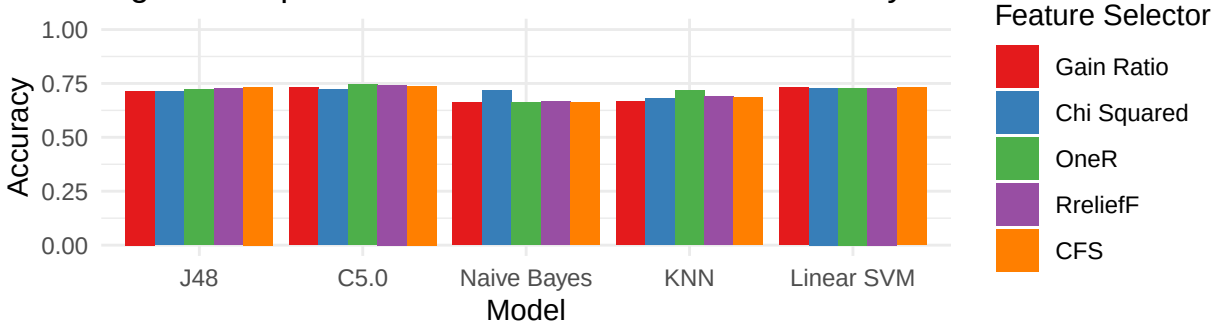


Fig. 6: Comparison of Model Performance: TPR

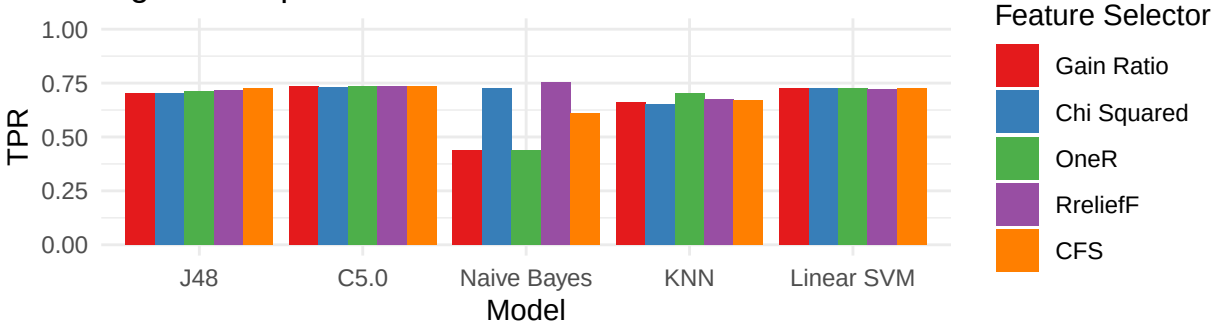


Fig. 7: Comparison of Model Performance: FPR

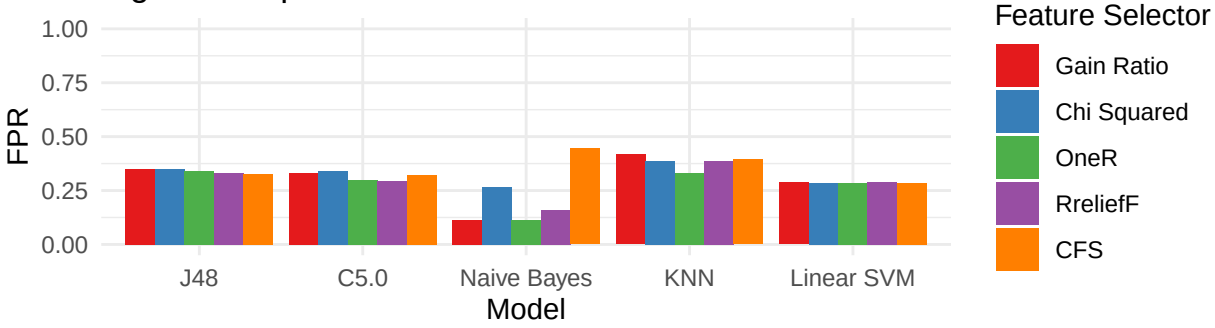


Fig. 8: Comparison of Model Performance: Precision

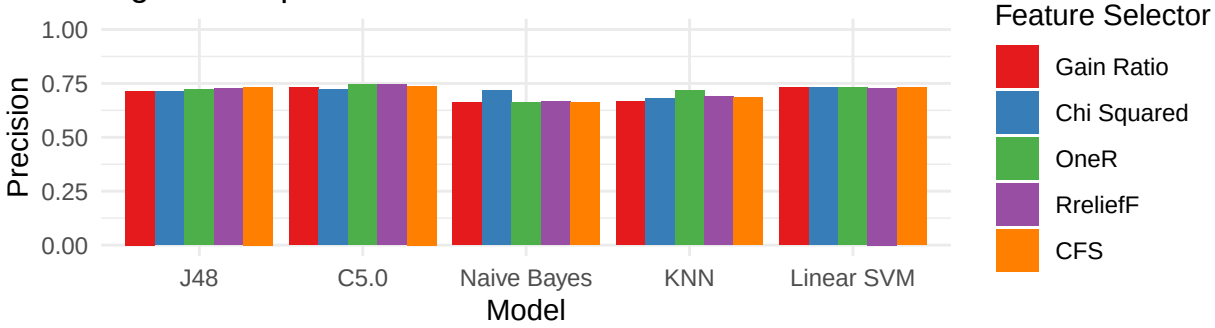


Fig. 9: Comparison of Model Performance: Recall

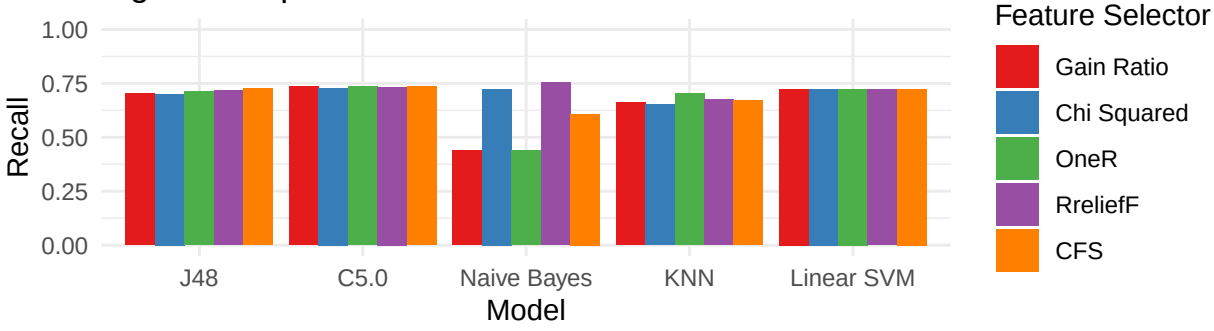


Fig. 10: Comparison of Model Performance: F1

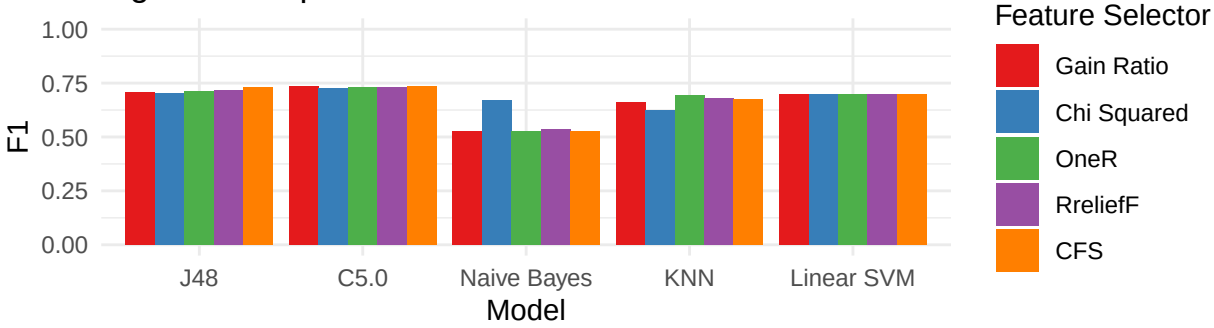


Fig. 11: Comparison of Model Performance: F2

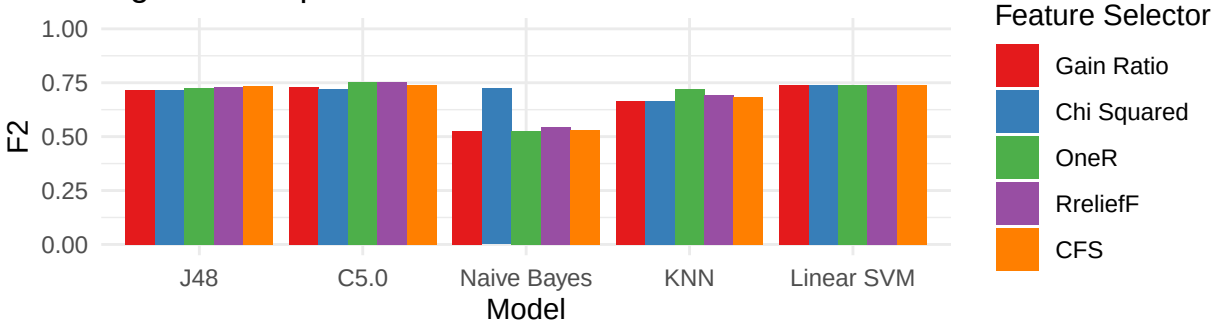


Fig. 12: Comparison of Model Performance: AUROC

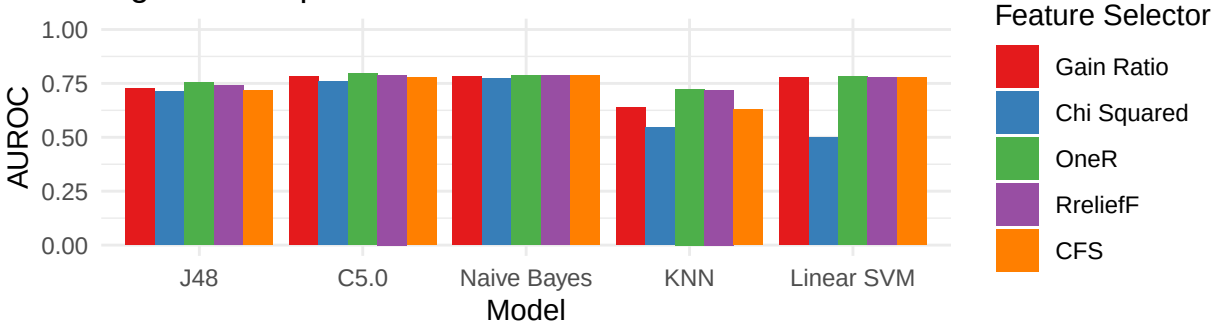
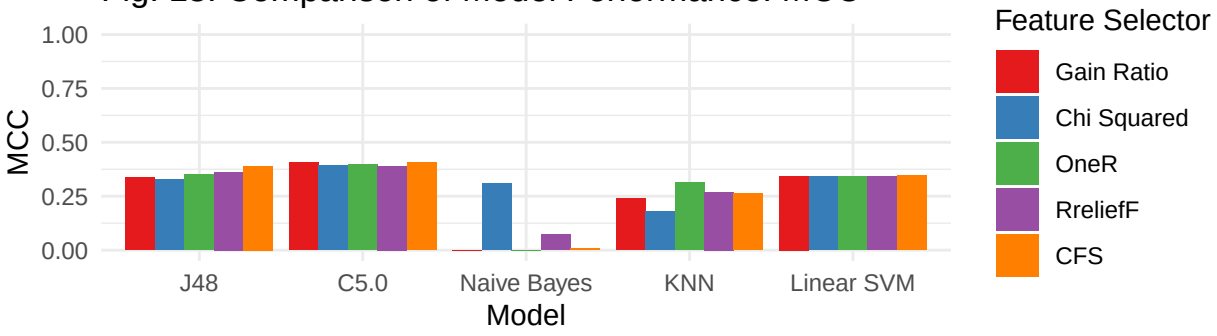


Fig. 13: Comparison of Model Performance: MCC



Appendix B: Model Results

Table 6: Confusion Matrix: J48 Algorithm, Gain Ratio Feature Selection

	Positive	Negative
Positive	667	455
Negative	702	2233

Table 7: Results: J48 Algorithm, Gain Ratio Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.715	0.594	0.239	0.487	0.594	0.536	0.569	0.727	0.336
Negative	0.715	0.761	0.406	0.831	0.761	0.794	0.794	0.727	0.336
Wtd. Avg.	0.715	0.705	0.349	0.715	0.705	0.707	0.718	0.727	0.336

Table 8: Confusion Matrix: J48 Algorithm, Chi Squared Feature Selection

	Positive	Negative
Positive	633	428
Negative	736	2260

Table 9: Results: J48 Algorithm, Chi Squared Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.713	0.597	0.246	0.462	0.597	0.521	0.564	0.712	0.326
Negative	0.713	0.754	0.403	0.841	0.754	0.795	0.795	0.712	0.326
Wtd. Avg.	0.713	0.701	0.350	0.713	0.701	0.703	0.717	0.712	0.326

Table 10: Confusion Matrix: J48 Algorithm, One-R Feature Selection

	Positive	Negative
Positive	680	442
Negative	689	2246

Table 11: Results: J48 Algorithm, One-R Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.721	0.606	0.235	0.497	0.606	0.546	0.581	0.754	0.351
Negative	0.721	0.765	0.394	0.836	0.765	0.799	0.799	0.754	0.351
Wtd. Avg.	0.721	0.712	0.340	0.721	0.712	0.714	0.725	0.754	0.351

Table 12: Confusion Matrix: J48 Algorithm, RreliefF Feature Selection

	Positive	Negative
Positive	674	412
Negative	695	2276

Table 13: Results: J48 Algorithm, RreliefF Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.727	0.621	0.234	0.492	0.621	0.549	0.590	0.739	0.362
Negative	0.727	0.766	0.379	0.847	0.766	0.804	0.804	0.739	0.362
Wtd. Avg.	0.727	0.717	0.330	0.727	0.717	0.718	0.732	0.739	0.362

Table 14: Confusion Matrix: J48 Algorithm, Correlation-based Feature Selection

	Positive	Negative
Positive	766	480
Negative	603	2208

Table 15: Results: J48 Algorithm, Correlation-based Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.733	0.615	0.215	0.560	0.615	0.586	0.603	0.718	0.39
Negative	0.733	0.785	0.385	0.821	0.785	0.803	0.803	0.718	0.39
Wtd. Avg.	0.733	0.728	0.328	0.733	0.728	0.730	0.736	0.718	0.39

Table 16: Confusion Matrix: C5.0 Algorithm, Gain Ratio Feature Selection

	Positive	Negative
Positive	840	555
Negative	529	2133

Table 17: Results: C5.0 Algorithm, Gain Ratio Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.733	0.602	0.199	0.614	0.602	0.608	0.604	0.784	0.405
Negative	0.733	0.801	0.398	0.794	0.801	0.797	0.797	0.784	0.405
Wtd. Avg.	0.733	0.734	0.331	0.733	0.734	0.733	0.732	0.784	0.405

Table 18: Confusion Matrix: C5.0 Algorithm, Chi Squared Feature Selection

	Positive	Negative
Positive	862	616
Negative	507	2072

Table 19: Results: C5.0 Algorithm, Chi Squared Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.723	0.583	0.197	0.630	0.583	0.606	0.592	0.758	0.394
Negative	0.723	0.803	0.417	0.771	0.803	0.787	0.787	0.758	0.394
Wtd. Avg.	0.723	0.729	0.342	0.723	0.729	0.726	0.721	0.758	0.394

Table 20: Confusion Matrix: C5.0 Algorithm, One-R Feature Selection

	Positive	Negative
Positive	668	336
Negative	701	2352

Table 21: Results: C5.0 Algorithm, One-R Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.744	0.665	0.230	0.488	0.665	0.563	0.620	0.795	0.398
Negative	0.744	0.770	0.335	0.875	0.770	0.819	0.819	0.795	0.398
Wtd. Avg.	0.744	0.735	0.299	0.744	0.735	0.733	0.752	0.795	0.398

Table 22: Confusion Matrix: C5.0 Algorithm, RreliefF Feature Selection

	Positive	Negative
Positive	633	306
Negative	736	2382

Table 23: Results: C5.0 Algorithm, RreliefF Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.743	0.674	0.236	0.462	0.674	0.549	0.618	0.789	0.391
Negative	0.743	0.764	0.326	0.886	0.764	0.821	0.821	0.789	0.391
Wtd. Avg.	0.743	0.734	0.296	0.743	0.734	0.729	0.752	0.789	0.391

Table 24: Confusion Matrix: C5.0 Algorithm, Correlation-based Feature Selection

	Positive	Negative
Positive	797	490
Negative	572	2198

Table 25: Results: C5.0 Algorithm, Correlation-based Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.738	0.619	0.206	0.582	0.619	0.600	0.611	0.777	0.406
Negative	0.738	0.794	0.381	0.818	0.794	0.805	0.805	0.777	0.406
Wtd. Avg.	0.738	0.735	0.322	0.738	0.735	0.736	0.740	0.777	0.406

Table 26: Confusion Matrix: Naive Bayes Algorithm, Gain Ratio Feature Selection

	Positive	Negative
Positive	0	0
Negative	1369	2688

Table 27: Results: Naive Bayes Algorithm, Gain Ratio Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.663	0.000	0.337	0.000	0.000	0.000	0.000	0.782	0
Negative	0.663	0.663	0.000	1.000	0.663	0.797	0.797	0.782	0
Wtd. Avg.	0.663	0.439	0.114	0.663	0.439	0.528	0.528	0.782	0

Table 28: Confusion Matrix: Naive Bayes Algorithm, Chi Squared Feature Selection

	Positive	Negative
Positive	349	121
Negative	1020	2567

Table 29: Results: Naive Bayes Algorithm, Chi Squared Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.719	0.743	0.284	0.255	0.743	0.380	0.537	0.774	0.31
Negative	0.719	0.716	0.257	0.955	0.716	0.818	0.818	0.774	0.31
Wtd. Avg.	0.719	0.725	0.267	0.719	0.725	0.670	0.723	0.774	0.31

Table 30: Confusion Matrix: Naive Bayes Algorithm, One-R Feature Selection

	Positive	Negative
Positive	0	0
Negative	1369	2688

Table 31: Results: Naive Bayes Algorithm, One-R Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.663	0.000	0.337	0.000	0.000	0.000	0.000	0.786	0
Negative	0.663	0.663	0.000	1.000	0.663	0.797	0.797	0.786	0
Wtd. Avg.	0.663	0.439	0.114	0.663	0.439	0.528	0.528	0.786	0

Table 32: Confusion Matrix: Naive Bayes Algorithm, RreliefF Feature Selection

	Positive	Negative
Positive	13	1
Negative	1356	2687

Table 33: Results: Naive Bayes Algorithm, RreliefF Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.666	0.929	0.335	0.009	0.929	0.019	0.046	0.785	0.074
Negative	0.666	0.665	0.071	1.000	0.665	0.798	0.798	0.785	0.074
Wtd. Avg.	0.666	0.754	0.161	0.666	0.754	0.535	0.544	0.785	0.074

Table 34: Confusion Matrix: Naive Bayes Algorithm, Correlation-based Feature Selection

	Positive	Negative
Positive	1	1
Negative	1368	2687

Table 35: Results: Naive Bayes Algorithm, Correlation-based Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.663	0.500	0.337	0.001	0.500	0.001	0.004	0.787	0.008
Negative	0.663	0.663	0.500	1.000	0.663	0.797	0.797	0.787	0.008
Wtd. Avg.	0.663	0.608	0.445	0.663	0.608	0.529	0.529	0.787	0.008

Table 36: Confusion Matrix: KNN Algorithm, Gain Ratio Feature Selection

	Positive	Negative
Positive	649	641
Negative	720	2047

Table 37: Results: KNN Algorithm, Gain Ratio Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.665	0.503	0.260	0.474	0.503	0.488	0.497	0.639	0.239
Negative	0.665	0.740	0.497	0.762	0.740	0.751	0.751	0.639	0.239
Wtd. Avg.	0.665	0.660	0.417	0.665	0.660	0.662	0.665	0.639	0.239

Table 38: Confusion Matrix: KNN Algorithm, Chi Squared Feature Selection

	Positive	Negative
Positive	270	202
Negative	1099	2486

Table 39: Results: KNN Algorithm, Chi Squared Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.679	0.572	0.307	0.197	0.572	0.293	0.414	0.546	0.18
Negative	0.679	0.693	0.428	0.925	0.693	0.793	0.793	0.546	0.18
Wtd. Avg.	0.679	0.652	0.387	0.679	0.652	0.624	0.665	0.546	0.18

Table 40: Confusion Matrix: KNN Algorithm, One-R Feature Selection

	Positive	Negative
Positive	511	294
Negative	858	2394

Table 41: Results: KNN Algorithm, One-R Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.716	0.635	0.264	0.373	0.635	0.470	0.557	0.724	0.313
Negative	0.716	0.736	0.365	0.891	0.736	0.806	0.806	0.724	0.313
Wtd. Avg.	0.716	0.702	0.331	0.716	0.702	0.693	0.722	0.724	0.313

Table 42: Confusion Matrix: KNN Algorithm, RreliefF Feature Selection

	Positive	Negative
Positive	587	477
Negative	782	2211

Table 43: Results: KNN Algorithm, RreliefF Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.69	0.552	0.261	0.429	0.552	0.483	0.522	0.72	0.27
Negative	0.69	0.739	0.448	0.823	0.739	0.778	0.778	0.72	0.27
Wtd. Avg.	0.69	0.676	0.385	0.690	0.676	0.679	0.692	0.72	0.27

Table 44: Confusion Matrix: KNN Algorithm, Correlation-based Feature Selection

	Positive	Negative
Positive	603	517
Negative	766	2171

Table 45: Results: KNN Algorithm, Correlation-based Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.684	0.538	0.261	0.440	0.538	0.485	0.515	0.63	0.262
Negative	0.684	0.739	0.462	0.808	0.739	0.772	0.772	0.63	0.262
Wtd. Avg.	0.684	0.671	0.394	0.684	0.671	0.675	0.685	0.63	0.262

Table 46: Confusion Matrix: Linear SVM, Gain Ratio Feature Selection

	Positive	Negative
Positive	474	202
Negative	895	2486

Table 47: Results: Linear SVM, Gain Ratio Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.73	0.701	0.265	0.346	0.701	0.464	0.582	0.775	0.344
Negative	0.73	0.735	0.299	0.925	0.735	0.819	0.819	0.775	0.344
Wtd. Avg.	0.73	0.724	0.287	0.730	0.724	0.699	0.739	0.775	0.344

Table 48: Confusion Matrix: Linear SVM, Chi Squared Feature Selection

	Positive	Negative
Positive	468	198
Negative	901	2490

Table 49: Results: Linear SVM, Chi Squared Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.729	0.703	0.266	0.342	0.703	0.460	0.580	0.498	0.342
Negative	0.729	0.734	0.297	0.926	0.734	0.819	0.819	0.498	0.342
Wtd. Avg.	0.729	0.724	0.287	0.729	0.724	0.698	0.739	0.498	0.342

Table 50: Confusion Matrix: Linear SVM, One-R Feature Selection

	Positive	Negative
Positive	470	199
Negative	899	2489

Table 51: Results: Linear SVM, One-R Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.729	0.703	0.265	0.343	0.703	0.461	0.581	0.783	0.343
Negative	0.729	0.735	0.297	0.926	0.735	0.819	0.819	0.783	0.343
Wtd. Avg.	0.729	0.724	0.287	0.729	0.724	0.698	0.739	0.783	0.343

Table 52: Confusion Matrix: Linear SVM, RreliefF Feature Selection

	Positive	Negative
Positive	468	200
Negative	901	2488

Table 53: Results: Linear SVM, RreliefF Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.729	0.701	0.266	0.342	0.701	0.459	0.579	0.776	0.341
Negative	0.729	0.734	0.299	0.926	0.734	0.819	0.819	0.776	0.341
Wtd. Avg.	0.729	0.723	0.288	0.729	0.723	0.698	0.738	0.776	0.341

Table 54: Confusion Matrix: Linear SVM, Correlation-based Feature Selection

	Positive	Negative
Positive	475	202
Negative	894	2486

Table 55: Results: Linear SVM, Correlation-based Feature Selection

Class	Accuracy	TPR	FPR	Precision	Recall	F1	F2	AUROC	MCC
Positive	0.73	0.702	0.264	0.347	0.702	0.464	0.583	0.777	0.345
Negative	0.73	0.736	0.298	0.925	0.736	0.819	0.819	0.777	0.345
Wtd. Avg.	0.73	0.724	0.287	0.730	0.724	0.700	0.739	0.777	0.345

Appendix C: Code

```
## Setup
# Import required libraries, installing any that are missing
# NB: This does not include all the packages that Caret would need
# to run the specified models, and there may be complexities to setting
# up a local environment to run them. For example, J48 requires Weka,
# a java configuration and the RWeka package.
required_libraries <- c(
  "data.table",
  "foreach",
  "FSelector",
  "rpart",
  "ModelMetrics",
  "doParallel",
  "caret",
  "ggplot2"
)
for (library_name in required_libraries) {
  if (!require(library_name, character.only = TRUE)) {
    install.packages(library_name, repos = "http://cran.us.r-project.org")
    library(library_name, character.only = TRUE)
  }
}
remove(library_name, required_libraries)

# Set plot theme for all figures
theme_set(theme_minimal())

# In order to make this entire report reproducible, I will set a seed manually
set.seed(2022)

# In order to expedite the analysis process,
# I'm going to leverage parallel computing. If running this yourself, be
# sure to change this to an appropriate number of threads for your hardware.
pCluster <- makePSOCKcluster(8)
registerDoParallel(pCluster)

## Data Preparation
# Read the raw data from file
data <- fread("project-2018-BRFSS-arthritis.csv", colClasses = "character")

# Save this for use later in the report content
raw.dims <- list(rows = nrow(data), cols = ncol(data))

# for the sake of R later, recode the class to Positive/Negative for arthritis
data[
  havarth3 == 1, havarth3 := "Positive"
][
  havarth3 == 2, havarth3 := "Negative"
][, havarth3 := factor(havarth3, levels=c("Positive", "Negative"))]

# For exploration, this will print cols with any amount of missing values
```

```r
# colSums(data == "?")[which(colSums(data == "?") > 0)]

# REMOVE columns from original sample design so they are not conflated with
# actual attributes
attrs.to.remove <- c(
  "x.psu",
  "x.llcpwt",
  "x.llcpwt2",
  "x.wt2rake",
  "x.ststr",
  "x.strwt",
  "x.rawrake"
)
for (col in attrs.to.remove) {
  data[, (col) := NULL]
}

# There are a few attributes that are duplicated by calculations, as I KNOW
# a priori from the codebook they are 100% correlated, I will remove one of the
# copies of the data here
for (col in c("weight2", "height3")) {
  data[, (col) := NULL]
}

# convert numeric columns
numericCols <- c(
  "drocdy3.",
  "htin4",
  "x.age80",
  "wtkg3",
  "x.bmi5",
  "htm4",
  "x.drnkwek"
)
for (col in numericCols) {
  data[,(col):=as.numeric(get(col))]
}

# Some of the columns require a little massaging here
# as discovered in the codebook
# Children: convert 88 => 0, 99 -> NA
data[children == 88, children := 0]
data[children == 99, children := NA]

# drocdy3: convert 900 to NA
data[drocdy3. == 900, drocdy3. := NA]
data[, drocdy3. := drocdy3. / 100]

# Divide wtkg3, x.bmi5, and htm4 by 100 to get implied decimals
data[, wtkg3 := wtkg3 / 100]
data[, x.bmi5 := x.bmi5 / 100]
data[, htm4 := htm4 / 100]
```

```
# drnkwek: convert 99900 to NA, divide by 100 to get implied decimals
data[x.drnkwek == 99900, x.drnkwek := NA]
data[, x.drnkwek := x.drnkwek / 100]

# recode ? in all other attributes as NA
for (col in setdiff(names(data), numericCols)) {
  data[data[[col]] == "?", (col):=NA]
  data[, (col) := factor(get(col))]
}

# Output a prescribed file. NB: that not all preprocessing has happened,
# as the imputation is dependent on splitting (so that imputation of
# test values is based on the train values alone).
write.table(
  data,
  "BrendanHawk-project-initial.csv",
  sep=",",
  row.names = FALSE
)

# Split the train and test datasets
trainingIndex <- createDataPartition(data$havarth3, p = 0.66, list = FALSE)
train <- data[trainingIndex,]
test <- data[-trainingIndex,]

# Perform sample imputation, use train data as informant
for (col in numericCols) {
  # get mean, sd from train data
  mu <- mean(train[[col]], na.rm = TRUE)
  sigma <- sd(train[[col]], na.rm = TRUE)

  # I want to make sure that integer-only attributes stay that way
  asInt <- (all.equal(as.integer(data[[col]]), data[[col]]) == TRUE)

  # apply imputation to both train and test sets
  train[is.na(get(col)), (col) := rnorm(.N, mu, sigma)]
  test[is.na(get(col)), (col) := rnorm(.N, mu, sigma)]

  # If these are integer-only attributes, make
  # sure to alter the random values accordingly
  if (asInt) {
    train[, (col) := ceiling(get(col))]
    test[, (col) := ceiling(get(col))]
  }

  # Also, center and scale this using the mu/sigma from train
  train[, (col) := (get(col) - mu)/sigma]
}

# Impute categorical values
for (col in setdiff(names(data), c(numericCols, "havarth3"))) {
  # Get distinct value counts from train data
  probs <- table(train[!is.na(get(col)), get(col)])
```

```
  # Impute in both sets by using the sample probs from above
  train[is.na(get(col)), (col) := sample(names(probs), .N, replace = TRUE, prob = probs)]
  test[is.na(get(col)), (col) := sample(names(probs), .N, replace = TRUE, prob = probs)]
}

# Assert that the train and test datasets have (roughly) the same class
# distributions as the original dataset
class.dist <- table(data$havarth3)/nrow(data)
train.dist <- table(train$havarth3)/nrow(train)
test.dist <- table(test$havarth3)/nrow(test)
# Checking the assertion to +/- 1% difference in class proportions
stopifnot(
  "Train dataset class distribution is not appropriate"=
      all.equal(train.dist, class.dist, tolerance = 0.01),
  "Test dataset class distribution is not appropriate"=
      all.equal(test.dist, class.dist, tolerance = 0.01)
)

# Write these datasets to file as prescribed
write.table(train, "BrendanHawk-project-training.csv", sep=",", row.names = FALSE)
write.table(test, "BrendanHawk-project-test.csv", sep=",", row.names = FALSE)

## Feature Selection
# There are 5 feature selection methods, but each will be run through
# the same evaluation process to choose a cutoff for "best" number of features.
# This helper function wraps that process into a single bit of code.
testFeatureSelectionMethod <- function(FSAttributes, dt) {
  # Run a series of tests using the k best features for
  # any number of k from 1 to all, gathering the AUC measure from a basic
  # CART algorithm using that subset of features.
  foreach(k = 1:nrow(FSAttributes), .combine = "c",
                  .packages=c("rpart", "FSelector", "ModelMetrics")) %dopar% {
    subset <- cutoff.k(FSAttributes, k)
    fit <- rpart(as.simple.formula(subset, "havarth3"), dt, method = "class")
    auc(fit)
  }
}

# For each of the types of attribute selection:
#   Calculate the measure
#   Benchmark
#   Take the "best" k values as determined by the elbow/scree plots in
#     Appendix A
attrs.gain.ratio <- gain.ratio(havarth3 ~ ., train)
attrs.gain.ratio.tests <- testFeatureSelectionMethod(attrs.gain.ratio, train)
attrs.gain.ratio.best <- cutoff.k(attrs.gain.ratio, 18)

attrs.chi.squared <- chi.squared(havarth3 ~ ., train)
attrs.chi.squared.tests <- testFeatureSelectionMethod(attrs.chi.squared, train)
attrs.chi.squared.best <- cutoff.k(attrs.chi.squared, 6)

attrs.oneR <- oneR(havarth3 ~ ., train)
attrs.oneR.tests <- testFeatureSelectionMethod(attrs.oneR, train)
```

```r
attrs.oneR.best <- cutoff.k(attrs.oneR, 12)

attrs.relief <- relief(havarth3 ~ ., train)
attrs.relief.tests <- testFeatureSelectionMethod(attrs.relief, train)
attrs.relief.best <- cutoff.k(attrs.relief, 21)

# The last does not get tested the same way, as the "best k" is somewhat
# built-in to the feature evaluation algorithm.
attrs.cfs.best <- cfs(havarth3 ~ ., train)

# A helper function to summarise the performance of a given classifier
summarise_performance <- function(probs, actual) {
  # Create predictions from class probabilities
  pred <- factor(
    ifelse(probs > 0.5, "Positive", "Negative"),
    levels = c("Positive", "Negative")
  )

  # Calculate the confusion matrix, making sure they're stored
  # as doubles so as not to mess up calculations later due to R's
  # handling of integers in strange ways sometimes.
  confmat <- as.double(caret::confusionMatrix(actual, pred)$table)

  # Summarise the confusion matrix values and total actual cases
  tp <- confmat[1]
  fn <- confmat[2]
  fp <- confmat[3]
  tn <- confmat[4]
  p <- sum(actual == "Positive")
  n <- sum(actual == "Negative")

  # The first stats are under the assumption that P is positive class
  TPR_1 <- (tp / (tp + fn))
  FPR_1 <- (fp / (fp + tn))
  Precision_1 <- (tp / (tp + fp))
  Recall_1 <- (tp / (tp + fn))
  F1_1 <- (2 * Precision_1 * Recall_1) / (Precision_1 + Recall_1)
  F2_1 <- ((1+2^2) * Precision_1 * Recall_1) / ((2^2 * Precision_1) + Recall_1)
  MCC_1 <- ((tp * tn) - (fp * fn)) / sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn))

  # in order to get the "Class 2" metrics, we flip P/N logically in the formulae
  TPR_2 <- (tn / (tn + fp))
  FPR_2 <- (fn / (fn + tp))
  Precision_2 <- (tn / (tn + fn))
  Recall_2 <- (tn / (tn + fp))
  F1_2 <- (2 * Precision_2 * Recall_2) / (Precision_2 + Recall_2)
  F2_2 <- (2 * Precision_2 * Recall_2) / (Precision_2 + Recall_2)
  MCC_2 <- ((tn * tp) - (fn * fp)) / sqrt((tn + fn) * (tn + fp) * (tp + fn) * (tp + fp))

  # Get an Accuracy value. There will be only one for this fit
  accuracy <- (tp + tn) / (p + n)
  # Get an AUROC value. There will be only one for this fit
  auroc <- auc(actual == "Positive", probs)
```

```r
  # Now format this all into a table to return to calling scope
  results <- data.frame(
    Accuracy = c(accuracy, accuracy),
    TPR = c(TPR_1, TPR_2),
    FPR = c(FPR_1, FPR_2),
    Precision = c(Precision_1, Precision_2),
    Recall = c(Recall_1, Recall_2),
    F1 = c(F1_1, F1_2),
    F2 = c(F2_1, F2_2),
    AUROC = c(auroc, auroc),
    MCC = c(MCC_1, MCC_2)
  )

  # To overcome some built-in math opinions of R, replace NA and NaN with 0
  results[is.na(results)] <- 0

  # Calculate a third row, the weighted averages of the other two
  results <- rbind(
    results,
    ((results[1,] * p) + (results[2,] * n)) / (p + n)
  )

  # Add class labels as a column
  results <- cbind(
    Class = c("Positive", "Negative", "Wtd. Avg."),
    results
  )

  # Return the confusion matrix and summary of performance measures
  return(list(
    confmat = data.frame(
      Positive = c(tp, fp),
      Negative = c(fn, tn),
      row.names = c("Positive", "Negative")
    ),
    summary = results
  ))
}

## Model experimentation
# This function wraps the process to train a model using the train dataset
# and a given attribute subset, then return a summary of its performance
# on the given test dataset using the same attribute subset
buildModelResults <- function(attrs, model, traindt, testdt, ...) {
  # Setting this seed here ensures that all training randomness is as
  # similar as possible. Importantly, this forces the cross validation
  # folds to always be the same in every model training process.
  set.seed(2022)
  model <- train(
    havarth3 ~ .,
    data = traindt[, c(attrs, "havarth3"), with = FALSE],
    method = model,
    trControl = trainControl(
```

```
      method = "cv",
      classProbs = TRUE,
      summaryFunction = twoClassSummary,
      seeds = NA
    ),
    metric = "ROC",
    ...
  )

  # The performance results of the outcomes predicted on the test set
  results <- summarise_performance(
    predict(
      model,
      testdt[, c(attrs, "havarth3"), with = FALSE],
      type="prob"
    )$Positive,
    testdt$havarth3
  )

  # Also return the model, so that we have access to the CV results later
  results$model <- model

  results
}

# Basic tree - J48
j48.gain.ratio.results <- buildModelResults(
  attrs.gain.ratio.best,
  "J48",
  train,
  test
)
j48.chi.squared.results <- buildModelResults(
  attrs.chi.squared.best,
  "J48",
  train,
  test
)
j48.oneR.results <- buildModelResults(
  attrs.oneR.best,
  "J48",
  train,
  test
)
j48.relief.results <- buildModelResults(
  attrs.relief.best,
  "J48",
  train,
  test
)
j48.cfs.results <- buildModelResults(
  attrs.cfs.best,
  "J48",
```

```
  train,
  test
)

# C5.0 - C5.0
# This is the only manually created tuning grid. This is done here
# so that "winnow" will always be false and model will always be "tree"
# instead of "rules". The trials values loosely mimic what would have been
# the defaults if no tuning grid was provided to caret, and corresponds to
# the boosting properties of this algorithm
C5.0.tuning <- data.frame(
  trials = seq(15, 95, 10),
  model = "tree",
  winnow = FALSE
)
C5.0.gain.ratio.results <- buildModelResults(
  attrs.gain.ratio.best,
  "C5.0",
  train,
  test,
  tuneGrid = C5.0.tuning
)
C5.0.chi.squared.results <- buildModelResults(
  attrs.chi.squared.best,
  "C5.0",
  train,
  test,
  tuneGrid = C5.0.tuning
)
C5.0.oneR.results <- buildModelResults(
  attrs.oneR.best,
  "C5.0",
  train,
  test,
  tuneGrid = C5.0.tuning
)
C5.0.relief.results <- buildModelResults(
  attrs.relief.best,
  "C5.0",
  train,
  test,
  tuneGrid = C5.0.tuning
)
C5.0.cfs.results <- buildModelResults(
  attrs.cfs.best,
  "C5.0",
  train,
  test,
  tuneGrid = C5.0.tuning
)

# Naive Bayes - nb
nb.gain.ratio.results <- buildModelResults(
```

```
  attrs.gain.ratio.best,
  "nb",
  train,
  test
)
nb.chi.squared.results <- buildModelResults(
  attrs.chi.squared.best,
  "nb",
  train,
  test
)
nb.oneR.results <- buildModelResults(
  attrs.oneR.best,
  "nb",
  train,
  test
)
nb.relief.results <- buildModelResults(
  attrs.relief.best,
  "nb",
  train,
  test
)
nb.cfs.results <- buildModelResults(
  attrs.cfs.best,
  "nb",
  train,
  test
)

# KNN - knn
knn.gain.ratio.results <- buildModelResults(
  attrs.gain.ratio.best,
  "kknn",
  train,
  test
)
knn.chi.squared.results <- buildModelResults(
  attrs.chi.squared.best,
  "kknn",
  train,
  test
)
knn.oneR.results <- buildModelResults(
  attrs.oneR.best,
  "kknn",
  train,
  test
)
knn.relief.results <- buildModelResults(
  attrs.relief.best,
  "kknn",
  train,
```

```
  test
)
knn.cfs.results <- buildModelResults(
  attrs.cfs.best,
  "kknn",
  train,
  test
)

# Linear SVM - svmLinear
lsvm.gain.ratio.results <- buildModelResults(
  attrs.gain.ratio.best,
  "svmLinear",
  train,
  test
)
lsvm.chi.squared.results <- buildModelResults(
  attrs.chi.squared.best,
  "svmLinear",
  train,
  test
)
lsvm.oneR.results <- buildModelResults(
  attrs.oneR.best,
  "svmLinear",
  train,
  test
)
lsvm.relief.results <- buildModelResults(
  attrs.relief.best,
  "svmLinear",
  train,
  test
)
lsvm.cfs.results <- buildModelResults(
  attrs.cfs.best,
  "svmLinear",
  train,
  test
)

# Stop the Parallel Cluster
stopCluster(pCluster)

# Print out the most commonly selected attributes
attr.selection.counts <- table(c(
  attrs.gain.ratio.best,
  attrs.chi.squared.best,
  attrs.oneR.best,
  attrs.relief.best,
  attrs.cfs.best
))
knitr::kable(
```

```
  attr.selection.counts[order(-attr.selection.counts)][1:5],
  col.names = c("Attribute", "Num. Times Selected"),
  caption = "Top 5 Chosen Attributes"
)

# labels to be used for output below
fselectorLabels = list(
  gain.ratio = "Gain Ratio",
  chi.squared = "Chi Squared",
  oneR = "OneR",
  relief = "RreliefF",
  cfs = "CFS"
)
# Gather all the required data for the AOV
aov.data <- data.frame(fselector = c(), AUROC = c())
for (fselector in c("gain.ratio", "oneR", "relief")) {
  results <- get(sprintf("C5.0.%s.results", fselector))
  aov.data <- rbind(
    aov.data,
    data.frame(
      `Feature.Selector` = sprintf("%s", fselectorLabels[[fselector]]),
      AUROC = results$model$results$ROC
    )
  )
}

# Print a one-way ANOVA
knitr::kable(
  summary(aov(AUROC ~ Feature.Selector, aov.data))[[1]],
  caption = "AOV: Cross-Validated AUROC Values vs Feature Selection Method",
  digits = 9
)

# Print adjusted pairwise t-tests
knitr::kable(
  TukeyHSD(
    aov(AUROC ~ Feature.Selector, aov.data),
    p.adjust.method = "bonferonni"
  )$Feature.Selector[,c(1,4)],
  caption = "Comparison of Means with Bonferonni Adjustment",
  digits = 5
)

# As prescribed, save the reduced test and training datasets from the
# final model in their own files.
write.table(
  train[, c(attrs.oneR.best, "havarth3"), with = FALSE],
  "BrendanHawk-best-train.csv",
  sep = ",",
  row.names = FALSE
)
write.table(
  test[, c(attrs.oneR.best, "havarth3"), with = FALSE],
```

```
  "BrendanHawk-best-test.csv",
  sep = ",",
  row.names = FALSE
)

# This function plots a scree chart of the benchmark results to evaluate
# and choose "Best" k feature sets
plotFeatureEval <- function(results, elbow, measure, figNum) {
  ggplot(
    data.frame(y = results, x = 1:length(results)),
    aes(x = x, y = y)
  ) + geom_vline(
    xintercept = elbow,
    col="red"
  ) + geom_line(
   #
  ) + labs (
    x = "Num. 'Best' Features",
    y = "AUROC",
    title = sprintf(
      "Fig. %d: Feature Evaluation Results %s",
      figNum,
      measure
    )
  )
}

# These have been manually evaluated for something
# similar to an elbow method. The goal is the fewest features with the
# highest AUC score from above, allowing for some margin of "good enough".
plotFeatureEval(attrs.gain.ratio.tests, 18, "attrs.gain.ratio.tests", 1) # 18
plotFeatureEval(attrs.chi.squared.tests, 6, "attrs.chi.squared.tests", 2) # 6
plotFeatureEval(attrs.oneR.tests, 12, "attrs.oneR.tests", 3) # 12
plotFeatureEval(attrs.relief.tests, 21, "attrs.relief.tests", 4) # 21

# The following for loops are a lot of R magic to repackage all of the
# performance evaluations for use plotting bar charts for more direct
# comparison between all models and feature selection approaches.
modelLabels = list(
  j48 = "J48",
  C5.0 = "C5.0",
  nb = "Naive Bayes",
  knn = "KNN",
  lsvm = "Linear SVM"
)
fselectorLabels = list(
  gain.ratio = "Gain Ratio",
  chi.squared = "Chi Squared",
  oneR = "OneR",
  relief = "RreliefF",
  cfs = "CFS"
)
for (modelName in c("j48", "nb", "C5.0", "knn", "lsvm")) {
```

```
  for (fselector in c("gain.ratio", "chi.squared", "oneR", "relief", "cfs")) {
    results <- get(sprintf("%s.%s.results", modelName, fselector))

    for (measure in names(results$summary)[2:ncol(results$summary)]) {
      measureResults <- dynGet(
        measure,
        ifnotfound = data.table(
          Model = c(),
          Features = c(),
          value = c()
        ),
        -1
      )

      measureResults <- rbind(
        measureResults,
        data.frame(
          Model = modelLabels[[modelName]],
          Features = fselectorLabels[[fselector]],
          value = results$summary[3, measure]
        )
      )

      assign(measure, measureResults)
    }
  }
}

# This function plots a grouped bar chart of a given performance measure
# for all models and attribute selection methods
plotModelEvaluation <- function(results, evalMetric, figNum) {
  ggplot(
    results,
    aes(
      x = ordered(Model, levels = unlist(modelLabels)),
      fill = ordered(Features, levels = unlist(fselectorLabels)),
      y = value)
  ) + geom_bar (
    position="dodge",
    stat="identity"
  ) + ylim (
    0, 1
  ) + labs (
    title = sprintf(
      "Fig. %d: Comparison of Model Performance: %s",
      figNum,
      evalMetric
    ),
    x = "Model",
    y = evalMetric,
    fill = "Feature Selector"
  ) + scale_fill_brewer(palette = "Set1")
}
```

```
# Plot all the performance evaluation measures individually
# With groups of the same model, color coded by feature selection type.
plotModelEvaluation(Accuracy, "Accuracy", 5)
plotModelEvaluation(TPR, "TPR", 6)
plotModelEvaluation(FPR, "FPR", 7)
plotModelEvaluation(Precision, "Precision", 8)
plotModelEvaluation(Recall, "Recall", 9)
plotModelEvaluation(F1, "F1", 10)
plotModelEvaluation(F2, "F2", 11)
plotModelEvaluation(AUROC, "AUROC", 12)
plotModelEvaluation(MCC, "MCC", 13)

# Print all confusion matrices and results summaries
knitr::kable(
  j48.gain.ratio.results$confmat,
  caption = "Confusion Matrix: J48 Algorithm, Gain Ratio Feature Selection"
)
knitr::kable(
  j48.gain.ratio.results$summary,
  caption = "Results: J48 Algorithm, Gain Ratio Feature Selection"
)
knitr::kable(
  j48.chi.squared.results$confmat,
  caption = "Confusion Matrix: J48 Algorithm, Chi Squared Feature Selection"
)
knitr::kable(
  j48.chi.squared.results$summary,
  caption = "Results: J48 Algorithm, Chi Squared Feature Selection"
)
knitr::kable(
  j48.oneR.results$confmat,
  caption = "Confusion Matrix: J48 Algorithm, One-R Feature Selection"
)
knitr::kable(
  j48.oneR.results$summary,
  caption = "Results: J48 Algorithm, One-R Feature Selection"
)
knitr::kable(
  j48.relief.results$confmat,
  caption = "Confusion Matrix: J48 Algorithm, RreliefF Feature Selection"
)
knitr::kable(
  j48.relief.results$summary,
  caption = "Results: J48 Algorithm, RreliefF Feature Selection"
)
knitr::kable(
  j48.cfs.results$confmat,
  caption = "Confusion Matrix: J48 Algorithm, Correlation-based Feature Selection"
)
knitr::kable(
  j48.cfs.results$summary,
  caption = "Results: J48 Algorithm, Correlation-based Feature Selection"
)
```

```
knitr::kable(
  C5.0.gain.ratio.results$confmat,
  caption = "Confusion Matrix: C5.0 Algorithm, Gain Ratio Feature Selection"
)
knitr::kable(
  C5.0.gain.ratio.results$summary,
  caption = "Results: C5.0 Algorithm, Gain Ratio Feature Selection"
)
knitr::kable(
  C5.0.chi.squared.results$confmat,
  caption = "Confusion Matrix: C5.0 Algorithm, Chi Squared Feature Selection"
)
knitr::kable(
  C5.0.chi.squared.results$summary,
  caption = "Results: C5.0 Algorithm, Chi Squared Feature Selection"
)
knitr::kable(
  C5.0.oneR.results$confmat,
  caption = "Confusion Matrix: C5.0 Algorithm, One-R Feature Selection"
)
knitr::kable(
  C5.0.oneR.results$summary,
  caption = "Results: C5.0 Algorithm, One-R Feature Selection"
)
knitr::kable(
  C5.0.relief.results$confmat,
  caption = "Confusion Matrix: C5.0 Algorithm, RreliefF Feature Selection"
)
knitr::kable(
  C5.0.relief.results$summary,
  caption = "Results: C5.0 Algorithm, RreliefF Feature Selection"
)
knitr::kable(
  C5.0.cfs.results$confmat,
  caption = "Confusion Matrix: C5.0 Algorithm, Correlation-based Feature Selection"
)
knitr::kable(
  C5.0.cfs.results$summary,
  caption = "Results: C5.0 Algorithm, Correlation-based Feature Selection"
)
knitr::kable(
  nb.gain.ratio.results$confmat,
  caption = "Confusion Matrix: Naive Bayes Algorithm, Gain Ratio Feature Selection"
)
knitr::kable(
  nb.gain.ratio.results$summary,
  caption = "Results: Naive Bayes Algorithm, Gain Ratio Feature Selection"
)
knitr::kable(
  nb.chi.squared.results$confmat,
  caption = "Confusion Matrix: Naive Bayes Algorithm, Chi Squared Feature Selection"
)
knitr::kable(
```

```
  nb.chi.squared.results$summary,
  caption = "Results: Naive Bayes Algorithm, Chi Squared Feature Selection"
)
knitr::kable(
  nb.oneR.results$confmat,
  caption = "Confusion Matrix: Naive Bayes Algorithm, One-R Feature Selection"
)
knitr::kable(
  nb.oneR.results$summary,
  caption = "Results: Naive Bayes Algorithm, One-R Feature Selection"
)
knitr::kable(
  nb.relief.results$confmat,
  caption = "Confusion Matrix: Naive Bayes Algorithm, RreliefF Feature Selection"
)
knitr::kable(
  nb.relief.results$summary,
  caption = "Results: Naive Bayes Algorithm, RreliefF Feature Selection"
)
knitr::kable(
  nb.cfs.results$confmat,
  caption = "Confusion Matrix: Naive Bayes Algorithm, Correlation-based Feature Selection"
)
knitr::kable(
  nb.cfs.results$summary,
  caption = "Results: Naive Bayes Algorithm, Correlation-based Feature Selection"
)
knitr::kable(
  knn.gain.ratio.results$confmat,
  caption = "Confusion Matrix: KNN Algorithm, Gain Ratio Feature Selection"
)
knitr::kable(
  knn.gain.ratio.results$summary,
  caption = "Results: KNN Algorithm, Gain Ratio Feature Selection"
)
knitr::kable(
  knn.chi.squared.results$confmat,
  caption = "Confusion Matrix: KNN Algorithm, Chi Squared Feature Selection"
)
knitr::kable(
  knn.chi.squared.results$summary,
  caption = "Results: KNN Algorithm, Chi Squared Feature Selection"
)
knitr::kable(
  knn.oneR.results$confmat,
  caption = "Confusion Matrix: KNN Algorithm, One-R Feature Selection"
)
knitr::kable(
  knn.oneR.results$summary,
  caption = "Results: KNN Algorithm, One-R Feature Selection"
)
knitr::kable(
  knn.relief.results$confmat,
```

```
  caption = "Confusion Matrix: KNN Algorithm, RreliefF Feature Selection"
)
knitr::kable(
  knn.relief.results$summary,
  caption = "Results: KNN Algorithm, RreliefF Feature Selection"
)
knitr::kable(
  knn.cfs.results$confmat,
  caption = "Confusion Matrix: KNN Algorithm, Correlation-based Feature Selection"
)
knitr::kable(
  knn.cfs.results$summary,
  caption = "Results: KNN Algorithm, Correlation-based Feature Selection"
)
knitr::kable(
  lsvm.gain.ratio.results$confmat,
  caption = "Confusion Matrix: Linear SVM, Gain Ratio Feature Selection"
)
knitr::kable(
  lsvm.gain.ratio.results$summary,
  caption = "Results: Linear SVM, Gain Ratio Feature Selection"
)
knitr::kable(
  lsvm.chi.squared.results$confmat,
  caption = "Confusion Matrix: Linear SVM, Chi Squared Feature Selection"
)
knitr::kable(
  lsvm.chi.squared.results$summary,
  caption = "Results: Linear SVM, Chi Squared Feature Selection"
)
knitr::kable(
  lsvm.oneR.results$confmat,
  caption = "Confusion Matrix: Linear SVM, One-R Feature Selection"
)
knitr::kable(
  lsvm.oneR.results$summary,
  caption = "Results: Linear SVM, One-R Feature Selection"
)
knitr::kable(
  lsvm.relief.results$confmat,
  caption = "Confusion Matrix: Linear SVM, RreliefF Feature Selection"
)
knitr::kable(
  lsvm.relief.results$summary,
  caption = "Results: Linear SVM, RreliefF Feature Selection"
)
knitr::kable(
  lsvm.cfs.results$confmat,
  caption = "Confusion Matrix: Linear SVM, Correlation-based Feature Selection"
)
knitr::kable(
  lsvm.cfs.results$summary,
  caption = "Results: Linear SVM, Correlation-based Feature Selection"
```

```
)
```