Supervised Machine Learning: Random Forests and Support Vector Machines in Classification Domains

Andrew P. DeVoss
Jayson C. Garrison

The University of Tulsa
Tandy School of Computer Science
CS-5333-01 (22/SP)
February 25, 2022

`https://github.com/jayson-garrison/ML-RandomForests-SVM`

Abstract

This project tackles four supervised learning classification tasks. Two of the domains are constructed with artificial data; the blobs data set includes two linearly separable normal clusters, and the spiral data set contains two differently labeled spirals that weave around each other. The remaining two domains contain real world data; the email data set gives the number of each word in many emails labeled either spam or not-spam, and the image data set contains pixel values for hand drawn numbers labeled 0 through 9. Here we present two approaches to these classification task: Random Forest and Support Vector Machines. We show that Random Forest is able to achieve a high accuracy for the blobs, email, and image domains, but fails to successfully partition the spiral data set in a generalizable manner. Subsequently, we show that a simplified Support Vector Machine is able to perform perfectly on both of the artificial data sets and perform well on the email data set given sufficient hyperparameter tuning.

1 Introduction

1.1 Supervised Learning

Supervised learning is the classical task to learn an inductive function $h : X \to Y$ from a hypothesis space H, where X is an input space with d dimensions and Y is an output space, given a set of labeled points $D \subseteq X$. Within supervised learning exists the problem of classification: if $|Y|$ is finite and small, that is to say there are few total labels for every $x \in X$, then the supervised learning problem can be expressed as learning the inductive h which best *classifies* X. For this project, we implement two famous approaches to learning an inductive classifier.

1.2 Random Forest

Random Forest is an ensemble technique to approximate the best $h \in H$. First proposed in 1995 [1], the general idea is to construct many weak Decision Trees, and treat the classification of the forest as some aggregate of the classifications of the trees in the forest, namely the mode. It can be shown that such an ensemble approach reduces the bias as well as the variance of the model, the latter of which needs only the application of the Central Limit Theorem of Statistics. The details of the implementation are presented in further sections.

1.3 Support Vector Machines

A Support Vector Machine (SVM) constructs a model that classifies a labeled space using a hyperplane as a decision surface. SVMs are often used to classify in domains where the label $y_i \in \{-1, 1\}$. Fundamentally, the SVM classifier relies on the solution to a linear optimization problem using a Lagrangian interpretation. Sequential Minimal Optimization (SMO) is one common way to interpret the fundamental optimization problem which iteratively tunes parameters in the minimal decomposition but does not guarantee the convergence to the global minimum. The Sequential Minimal Optimization approach leaves room for hyperparameter experimentation where a violation tolerance and loss function penalty constant can be varied. Furthermore, data that is not linearly separable can be mapped and classified into higher dimensional space with the use of Kernel functions.

2 Related Work

Random Forests and Support Vector Machines are still widely used in industry, research, and data analysis. One common issue that all classifiers encounter is a trade off, pertaining to offline learning, between model efficiency and running time complexity. Since Random Forests and SVMs are both commonly used in binary classification they are often compared against one another regarding their bias-variance trade off tendencies and effectiveness on certain data sets. Random Forests leave room for improvement in the information gain functions which contribute to algorithmic intensity. One example for recent improvement is the Fujiwara *et al.* F-forest proposition [2] that incrementally estimates upper bounds relating to impurity calculations in finding the optimal split point. The F-forest model classified several data sets including MNIST with an accuracy competitive with the Breiman *et al.* *randomForest* [3], but also the *Rborist* [4] and *ranger* [5] implementations which are modern highly efficient adaptations of the Random Forest learner. In addition to this, the F-forest model is 144 times faster than the *randomForest* and over 2 times faster than the modern Random Forest adaptations. The Fujiwara *et al.* proposition supports that the Random Forest model is continually improving and adapting to large data sets were runtime increases. Our implementation of Random Forests topically applies the basic model to analyze the general efficiency of Decision Tree ensemble learners.

The Sequential Minimal Optimization algorithm can be expensive in terms of time complexity. In order to overcome this, Wen *et al.* [6] proposes the utilization of multi-core CPUs and high performance GPUs to significantly speed up SVM implementations including the

SMO algorithm. Using hyper-efficient parallelism routines, Wen *et al.* proposed Thunder-SVM which successfully sped up Chang and Lin's LibSVM [7]. One data set that took over one week to train and test was sped up to only 1251 seconds using GPU acceleration. The Wen *et al.* ThunderSVM is generally 10 times faster than using LibSVM without multi-core CPU and GPU speed-up techniques. This is relevant for our work as, by implementing the SMO algorithm, SVMs can take a long time to fit a data set if the worst case complexity is reached.

3 Data Sets Used

In order to analyze the effectiveness of Random Forests learners and SVM implementations, we classified labels on different data sets. Training and validating the two different supervised machine learning techniques on the same sets of data allows for the direct comparison of classification accuracy. Two artificial (contrived) data sets were used along with two realistic data sets representing actual classification problem domains.

3.1 Artificial

The first of the two artificial data sets consisted of linearly separable clusters of points with two labels. This problem represents a trivial domain in order to verify that each of our models can classify linearly separable data. This baseline is helpful when training the Random Forest and SVM on other data sets where noise is present to determine if they were correctly classifying a simpler data set.

The other contrived data set was also a two label classification domain but the points were intermixed in a spiral making the data not linearly separable. This artificial domain represented more realistic problems where data is not linearly separable requiring a either a feature construction technique, such as Principal Component Analysis, or a dimension expansion.

3.2 Realistic

The first of the two real world data sets was a partition of the MNIST repository [8] and consisted of 42,000 28 pixel by 28 pixel images of handwritten digits labeled $\{0, 1, 2, \ldots 9\}$. We simplified the features by defining them as pixel intensity values ranging from 0 to 255 instead of pixel location.

The latter realistic data set consisted of emails [9] with English words that were vectorized such that each email data point had the same set of possible features where repeat words are considered. Compared to the Naïve Bayes classifier [10], this data set proves difficult for the Random Forest learner as most features in the email data set are not correlated with the label.

3.3 Use of Principal Component Analysis

In the case of both the image and email data sets there are several features that contribute little information to the associated classifications. For example, when building Decision Trees with the email data set it is highly probable that the sub-sample of attributes considered at each node yields minimal information gain, which results in the construction of an unhelpful learner in the Random Forest. If we left the email data set unaltered, the Random Forest would consist of a large majority of learners that classify instances based on features that have little correlation with the label. A similar problem also exists in the image data set for pixel intensities. To mitigate this difficulty, we performed Principal Component Analysis (PCA) on the two data sets and significantly reduced the feature dimension to consist of strong identifiers.

By utilizing 10 and 4 principal components for the image and email data sets respectively, both the Random Forest and SMO algorithms were able to classify with greatly increased accuracy.

4 Model Explanation and Implementation

In this section, further explanation is given for the intuition behind each heuristic. Lower level details of the two models are presented. We indicate for consideration any stylistic choices made in development that were optional or supplementary to the base project.

4.1 Random Forest

4.1.1 Decision Trees

In order to understand how a Random Forest works, it is first necessary to understand the constituent parts, namely Decision Trees. A single Decision Tree, very similar in concept to a decision flow chart, is a classification algorithm that utilizes an acyclic rooted tree structure to assign a label to a data instance. Each node in the branching tree represents a splitting on some combination of feature values. For example, the root node might branch on whether the first feature has a value less than 25.3 or a value greater than or equal to 25.3. The true feature value of the data instance being classified determines whether it continues down the left or right logical subtree. This process repeats until a terminal node, or “leaf node,” of the structure is reached.

Although it is relatively easy to understand how a Decision Tree classifies instances, the process to learn a Decision Tree recursively is more complex. Traditionally, the tree is constructed from the root node down, at each node selecting the best attribute, or combination thereof, for partitioning the data. For implementation purposes, the qualifier *best* needs to be more explicitly defined. One way to measure the quality of a partition given a set of data and possible attributes to split on is to measure information gain. This can be thought of as the node impurity before the split minus the node impurity after splitting; if impurity is high before the split, which is to say there is higher uncertainty about the answer, then there is more information in the answer. Thus, the attribute used at each node is the attribute that yields the partition which maximizes information gain for a collection of samples; this attribute is selected. Then, for each set of samples in the partition of the original collection, a recursive procedure is called using that subset.

Node impurity can be measured in several ways. For this project, we used three different functions to estimate the impurity for our information gain procedure. Below, these three functions are given as equations, where n is a node, l is a class with $l \in [1, k]$, and $p_l(n)$ denotes the fraction of points at node n that belong to class l.

$$\begin{aligned} Entropy &= \sum_{l=1}^{k} p_l(n) * (1 - p_l(n)) \\ Gini &= -\sum_{l=1}^{k} p_l(n) * log_2(p_l(n)) \\ Misclassification &= 1 - max_{l \in [1,k]} p_l(n) \end{aligned}$$

The tree building process can terminate in several ways. For example, if the subset of the samples being considered at a node in the subtree are homogeneous in classification, then there is no need to choose another attribute and further divide them. Certain regularization techniques also terminate the process early, such as enforcing a maximum number of total nodes in the tree, a maximum tree depth, or a minimum number of samples at each node. In our implementation, we chose to vary the depth of each Decision Tree in the Random Forest with a hyperparameter $M \in \{1, 2, 3\}$.

4.1.2 Random Forest Ensemble Learning

With an understanding of Decision Tree classification and construction, one can begin to understand the Random Forest approach to classification. Random Forest is an ensemble learning method, meaning it combines n inductive hypotheses $h_1, h_2, ..., h_n$ and uses a voting system to determine the output. In a Random Forest, each h_i is a learned Decision Tree classifier. As mentioned in the introduction, it can be shown that the aggregate of many weak learners reduces both the overall bias and the variance of the ensemble model. The next procedure to be considered is how to construct each of the trees in the forest.

Building a Decision Tree from a set of samples with a collection of attributes and a stable information gain measure is a deterministic procedure. If every tree was constructed with these held constant, then every tree in the forest would be the same. In order to introduce randomness (and improve the model), one employable technique is bootstrap aggregating, or bagging. With bagging, each tree learns on a different set of data instances constructed by sampling with replacement from the original data set. To introduce even more randomness, at each node only a small subset of the potential attributes are considered for splitting, represented in our implementation as a hyperparameter $M \in \{2, .5 * log_2(|A|), log_2(|A|)\}$, where $|A|$ is the total number of attributes.

Another option for increasing the randomness during an ensemble model construction process is known as boosting. In boosting, the training data used for subsequent models is a weighted selection of the original data, where preference is given to samples that were misclassified by the previous models. Additionally, the prediction of the ensemble is typically a weighted prediction of the constituents, giving preference to the "experts" among the group. While boosting is very common in practice, it requires sequential building and cannot be fully parallelized as bagging can. For our project, we elected to implement bagging only.

4.2 Support Vector Machines

4.2.1 The Fundamental Optimization Problem

The Support Vector Machine, similar to the Random Forest, is a discrete decision surface classifier. Fundamentally a linear optimization problem, the SVM constructs a linear surface for discrimination, and therefore performs well on data that is linearly separable or can be easily represented as such. The construction of a SVM, as described by Hsu *et al.*, requires the solution to

$$\begin{aligned} &min_{\mathbf{w},b,\xi} && \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{m}\xi_i \\ &\text{Subject to} && y_i(\mathbf{w}^T\phi(\mathbf{x}_i)+b) \geq 1-\xi_i \\ & && \xi_i \geq 0. \end{aligned}$$

Here, ϕ represents a transformation that allows the feature vector, $\mathbf{x}_i$, to be mapped in a higher dimension. The SVM, as presented in the Hsu *et al.* interpretation, finds a hyperplane where the margins between the plane and its support vectors are maximized. The slack variables ξ_i handle the case where data may not be perfectly linearly separable.

In order to build a functional SVM we must solve the fundamental linear optimization problem presented by Hsu *et al.*. One way to do this is to minimize the Lagrangian for all $\mathbf{w}, b, \alpha_i \geq 0, \beta_i \geq 0$ shown by

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i[y_i(\mathbf{w}\times\mathbf{x}_i + b) + \xi_i] - \sum_{i=1}^{m}\beta_i\xi_i.$$

Maintaining the slack variable ξ we introduce two constraint parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ as vectors to be optimized based on our training set $\mathbf{x}$ and the associated labels $\mathbf{y}$. Now, a set of constraints imposed for optimization known as the Karush-Kuhn-Tucker (KKT) conditions are

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i,$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0,$$

$$\alpha_i + \beta_i = C,$$

$$\forall i \in [1, m], \alpha_i[y_i(\mathbf{w} \times \mathbf{x}_i + b) + \xi_i] = 0,$$

$$\text{and} \qquad \beta_i \xi_i = 0.$$

With the KKT conditions we now have C, the strength of the margin, bounding parameters. Furthermore, we constrain $\mathbf{w}$ where, after optimization, it describes the vector result of each feature vector, optimized α, and associated label. By imposing the KKT conditions, specifically plugging in $\mathbf{w}$ into the Lagrangian form of the fundamental optimization problem, we obtain what is known as the dual problem.

$$L = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \mathbf{x}_j)$$

4.2.2 Sequential Minimal Optimization Interpretation

Due to the nature of the dual optimization problem, finding optimal values for $\boldsymbol{\alpha}$ to construct $\mathbf{w}$ can result in an algorithmically complex routine which often results in the use of external packages for the applications of SVM models. However, if we interpret the optimization process of $\boldsymbol{\alpha}$ iteratively by selecting and optimizing only two parameters α_i and α_j sequentially, we can solve the dual problem efficiently. This process, which tackles the minimal decomposition of the optimization problem, is fittingly called Sequential Minimal Optimization (SMO). SMO seeks to obtain a linear classifier of the form

$$\begin{aligned} f(\mathbf{x}) &= \boldsymbol{w}^T \mathbf{x} + b, \text{and equivalently} \\ &= \sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b. \end{aligned}$$

$K(\mathbf{x}_i, \mathbf{x})$, the Kernel Function, can replace the standard inner product to increase the dimensions of the data. In our implementation we define three Kernel Functions for comparison:

$$K(\mathbf{x}_i, \mathbf{x}) = \begin{cases} \langle \mathbf{x}_i | \mathbf{x} \rangle, & \text{The Standard Inner Product} \\ \exp(-\frac{||\mathbf{x}_i - \mathbf{x}||}{2\sigma^2}), & \text{The Gaussian Kernel} \\ (\mathbf{x}_i^T \mathbf{x} + c)^2, & \text{The Quadratic Kernel} \end{cases}$$

Kernels allow for expanding the feature dimensions of the data. By mapping data into higher dimensions, a linear decision surface may create a better partition. Kernels are especially helpful for data that is not linearly separable in its unaltered basis.

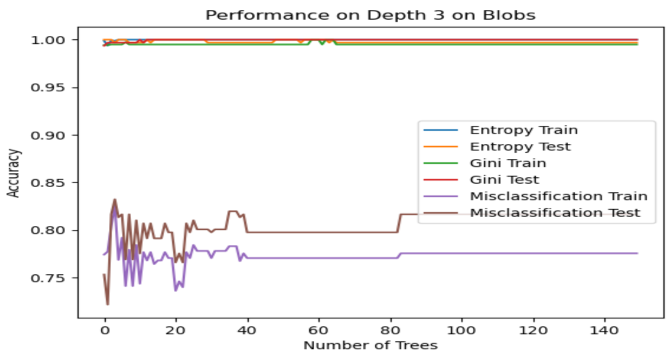
The simplified SMO algorithm iteratively selects α_i and α_j, optimizes them, and adjusts the b parameter. At each step, the KKT conditions are checked for violations within a tolerance, which we use as a hyper parameter along with the constant C. The solution to the optimization problem is reached when the α's converge.

5 Performance

5.1 Random Forest

5.1.1 Artificial Data Sets

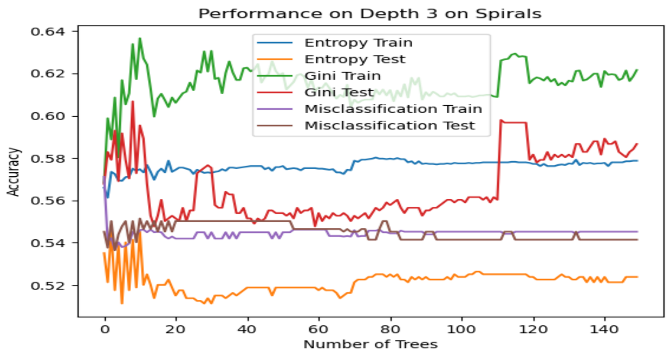
Figure 1:



The Random Forest learner performs well on linearly separable data sets as expected. Branching on x_1 or x_2 and defining the classifier threshold on a collection of weak learners is extremely effective. Based on Figure 1, the Entropy and Gini functions solidly classify the data near 100% training and testing accuracy with a maximum depth of 3 for any weak learner in the forest. However, the Misclassification function performs noticeably worse than the Entropy and Gini implementations. The increase of maximum depth did not significantly increase the accuracy of the Entropy and Gini measures, as they remained close to 100%, however, the increase in depth significantly improved the accuracy when using Misclassification by over 25%.

Compared to the Blob data set, the Spiral data is far harder to classify with the Random Forest learner. Since this ensemble classifier relies on discriminitve split points in the weak learners to make a decision, which is not easily determined in a non linearly separable data set, the training and testing accuracy using the Random Forest learner is worse. Figure 2 illustrates that all three functions performed within 4% testing accuracy of each other showing that the data set directly affects the performance of the Random Forest classifier.

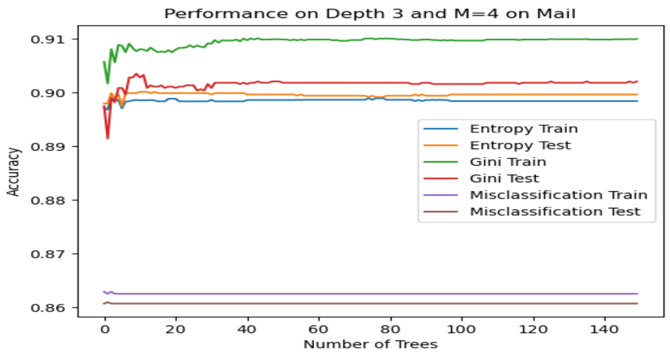
Figure 2:



Despite this, the Gini function yielded the best validation accuracy with a sharp increase when the number of trees in the forest reached 110.

5.1.2 Realistic Data Sets

Figure 3:

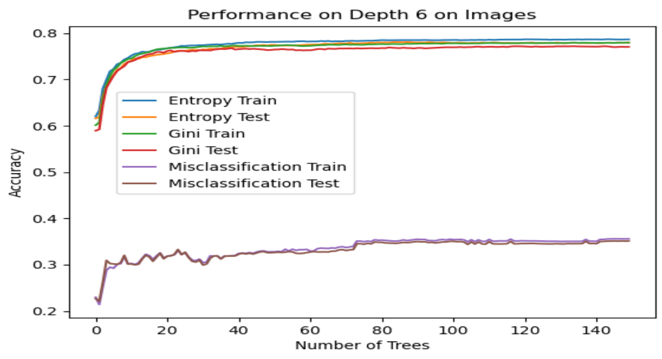


With the new email feature space constructed using PCA, the Random Forest learner produced strong results in all three information gain techniques. Most notably in Figure 3 is the significant difference between Gini and Entropy when compared with Misclassification. With Gini and Entropy implementations performing 4% better than the Misclassification measure, this suggests that Gini and Entropy discriminate features more efficiently and therefore extract more information on attribute split point selection.

Since the feature dimension was reduced to 4 in the email domain, the optimal value of M, the number of attributes to consider splitting at each node, yielded the best validation accuracy since it covers the whole feature space. If the number of attributes remained 3000, this would not be the case. We ran the Random Forest approach on an M value of 2 which achieved approximately the same results as $M = 4$ with Gini performing 4% better than both Misclassification and Entropy. Furthermore, the range in testing accuracies were worse for small forests. However, both M values of 2 and 4 across all three information gain measures converged at a forest size of around 40 trees.

Compared to the mail data set, images were harder to classify with the Random Forest. This

Figure 4:



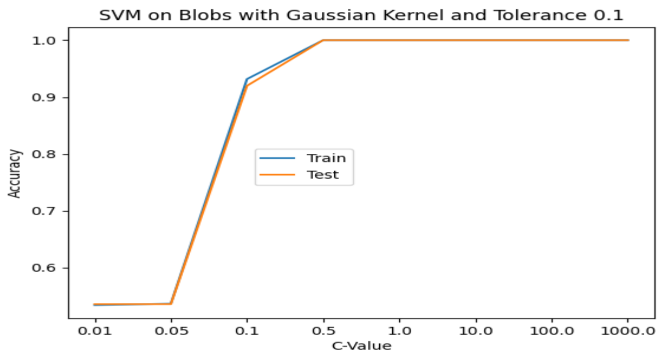
is expected since the classification of images are not boolean; rather they are labeled into ten different classes. Therefore, splitting on different attributes is harder as node impurity is not as decisive with more possible values in the label space. As a result of this difficulty, larger trees are required in the forest with a minimum depth of 4 in order to allow each learner to label each possible image classification. Here we also analyze a value of $M = 3$ which considers 3 out of 10 possible attributes at each node. Since the number of attributes were reduced to 10 using PCA, a larger value of M produces greater testing accuracy. We see that, in Figure 4, between depths of 4, 5, and 6, Random Forests with a depth of 6 provided the best results with Gini and Entropy converging to approximately 80% test accuracy after a forest of 25 trees. The convergence of Entropy and Gini information gain measures in the Random Forest implementation after 25 trees is significant due to the 20% increase from a forest beginning at 1 tree to over 25 trees suggesting that the ensemble method of several weak learners greatly increases test accuracy.

5.2 Support Vector Machine

5.2.1 Artificial Data Sets

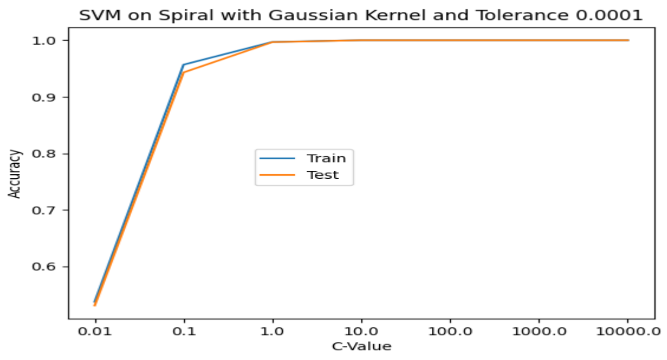
The linear separability of the data set greatly influenced the strictness required to achieve high classification performance. For the blob data set, which is fully linearly separable in the original feature space, a C-value of 0.5 and tolerance of 0.1 were sufficient to achieve a testing validation of 100% accuracy. Again, the constant C is the strength of the margin and serves as an upper bound for each α_i. Further, the tolerance value indicates how tightly the KKT-conditions must be satisfied before the optimization problem can be considered solved. The fact that a low value for C and high value for tolerance were sufficient to obtain perfect performance during cross validation indicate that the blob classification is easily solved by SVM.

Figure 5:



Use of the Gaussian Kernel yielded similar performance on the spiral data set. Without being transformed, the spiral data set is not linearly separable. However, the Gaussian Kernel transforms data into an infinite dimensional space. With a sufficiently high value for C and a sufficiently low tolerance, our SVM was again able to determine a hyper-plane which perfectly splits the data in the transformed space. The least strict combination of hyperparameters that achieved this performance were $C = 0.5$ and $Tolerance = 0.0001$ as can be seen by Figure 6. For the spiral classification task, the critical range of C-values appears to be $[0.01, 1]$.

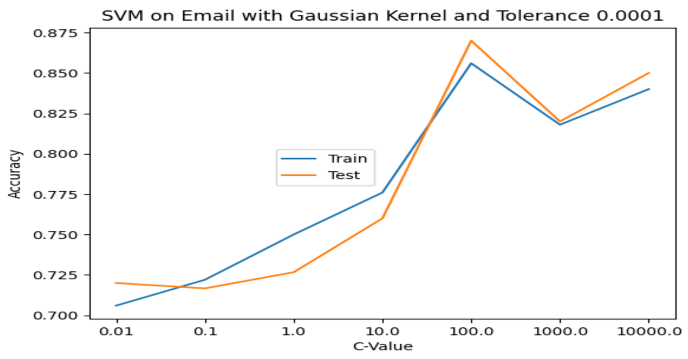
Figure 6:



5.2.2 Realistic Data Sets

The simplistic formulation of SVM aims to separate boolean data. As the image data set has ten different class labels, we do not present results for that domain. Figure 7 displays results in the email classification domain achieved using SVM with the Gaussian Kernel transformation and $Tolerance = 0.0001$. It shows that a C-value of well over 10 is necessary to achieve performance significantly better than always assigning the majority label. Values of C ranging higher than 100,000 are excluded as computation time significantly increased. Also note that our SVM was unable to perfectly separate the email domain with the use of the Gaussian Kernel. Even though every labeling of unique points is fully linearly-separable in infinite dimensional space, SMO is not guaranteed to converge, and it may be the case the our optimization routine was not sufficiently relaxed. In actuality, our SVM achieved the best performance when discriminating between email samples by using the Quadratic Kernel and the hyperparameter combination $C = 10,000, Tolerance = 0.0001$. With this combination, the average testing accuracy was 93.8%.

Figure 7:



6 Analysis

6.1 Model Algorithmic Complexity

The complexity of the Random Forest model is difficult to construct due to the large effect hyperparameters have on the computation. In general, our Random Forest implementation has an algorithmic complexity of $\mathcal{O}(poly(|\mathbf{x}|, |\mathbf{x}_i|, |\mathbf{x}_{i,j}|, M, T_d))$ where $|\mathbf{x}|$ is the size of the training instances, $|\mathbf{x}_i|$ is the number of attributes, $|\mathbf{x}_{i,j}|$ are the possible attribute values for the i^{th} attribute, M is the number of attribute split points considered at each node, and T_d is the maximum tree depth. This is compared to the SMO implementation which has a worst case complexity of $\mathcal{O}(|\mathbf{x}|^3)$ when all $\alpha_i \in \boldsymbol{\alpha}$ are considered for optimization. Both the SVM and Random Forest have the potential to reach similar complexities when 2 or more of the variables in the Random Forest model become similar to $|\mathbf{x}|$. We have observed that when more attributes are considered at each node when constructing the tree the time of the Random Forest fitting routine significantly increases. If the maximum depth of the tree were to also increase along with M the training process would increase even more. On the other hand, if the data set caused the SMO `fit()` function to not optimize all $\boldsymbol{\alpha}$ then the worst time complexity would not be reached, allowing for a run time comperable to the Random Forest leaner in a lower polynomial order.

6.2 Comparing Models

6.2.1 Artificial Data Sets

The first artificial domain requires constructing a decision surface to discriminate between two normal clusters of points that are in separable in two dimensions. Because of the innate separability, both Random Forest and SVM were able to achieve a perfect average cross validation testing accuracy of 100%. The weakness of some hyperparameter combinations show through on this data set: for example, the Misclassification node impurity measure was unable to score over 85% testing accuracy even with 150 trees in the forest.

The difference in performance between the models is most evident on the spiral data set. As each Decision Tree in the Random Forest is only able to make single-variable splitting decisions, every edge on the decision surface is axis parallel. Even though the spiral data has only two feature dimensions, the number of axis parallel line segments required to homogeneously partition the samples is large. Under the best hyper parameter combination we tested, Random Forest achieved less than a 65% test accuracy. On the other hand, SVM was able to learn an optimal decision surface for the spiral data with the use of the Gaussian Kernel, achieving a 100% test accuracy.

6.2.2 Realistic Data Sets

In the email domain, the maximum average test accuracy was comparable for Random Forest and for SVM. Both achieved an accuracy of over 90% and neither surpassed 95%. Compared to a baseline Naïve Bayes classifier that achieved upwards of 97% testing accuracy [10], each of these models fall short.

For classifying hand written images, the Random Forest model was able to achieve a testing accuracy of approximately 80% when using either Gini or Entropy to measure impurity. As there are 10 different labels, 80% accuracy is significantly higher than what could be achieved by random guessing based on the prior distribution. When using Misclassification, the model struggled to surpass an accuracy of 30%. We did not modify the problem to a boolean domain, so the simplified implementation of SVM is untested for this domain.

7 Conclusion

The Random Forest approach to modeling an inductive function for supervised learning forms a strong baseline for binary and n-ary classification domains. Random Forest was able to achieve high testing accuracy in each domain that required few discreet edges for an optimal decision surface. Similarly, SVM was able to achieve high accuracy on each of the binary domains, including the spiral artificial data, which required a high dimensional extension of the data to become linearly separable. Each of the models we have detailed form the backbone of competitively viable modern techniques. Our implementations and validation thereof support their viability on real world problem sets.

References

[1] T. K. Ho, “Random decision forests,” in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, 1995, pp. 278–282 vol.1.

[2] Y. Fujiwara, Y. Ida, S. Kanai, A. Kumagai, J. Arai, and N. Ueda, “Fast random forest algorithm via incremental upper bound,” in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, ser. CIKM ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2205–2208. [Online]. Available: https://doi.org/10.1145/3357384.3358092

[3] L. B. Statistics and L. Breiman, “Random forests,” in *Machine Learning*, 2001, pp. 5–32.

[4] M. Seligman, “Rborist package,” Available: https://cran.r-project.org/web/packages/Rborist/index.html, 2019.

[5] S. W. Marvin N. Wright and P. Probst, “Package ’ranger’, howpublished =.”

[6] Z. Wen, J. Shi, Q. Li, B. He, and J. Chen, “Thundersvm: A fast svm library on gpus and cpus,” *J. Mach. Learn. Res.*, vol. 19, no. 1, p. 797–801, jan 2018.

[7] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,” *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[8] “Digit recognizer.” [Online]. Available: https://www.kaggle.com/c/digit-recognizer/data?select=train.csv

[9] J. Vaidya, M. Kantarcıoğlu, and C. Clifton, “Privacy-preserving naïve bayes classification,” *The VLDB Journal*, vol. 17, no. 4, p. 879–898, jul 2008. [Online]. Available: https://doi.org/10.1007/s00778-006-0041-y

[10] J. C. Garrison, “Supervised machine learning: The naïve bayes classifier applied in the multivariate bernoulli event and multinomial event models in image and text data,” Available on GitHub:https://github.com/jayson-garrison/ML-Naive-Bayes/blob/main/CS_5333_Naive_Bayes_Classification_Report(1).pdf.