Random Forest & SVM Project - CS-5333

Rylee Buchert, Noah Hall

February 2022

Abstract

Support vector machines and random forests are two forms of supervised learning algorithms that exhibit strong performance in classification tasks. The former is a form of ensemble learning which combines the output of many decision trees to generate predictions. On the other hand, SVM's attempt to find the biggest 'gap' between data classes and generate a decision boundary that maximizes the separation. Both of these models have a wide variety of real-world applications, proving to be some of the most effective forms of 'off-the-shelf' supervised machine learning techniques. In this project, the machine learning models were implemented and tested on several datasets, both from real-life contexts and artificial domains. The models expressed high accuracy in classification problems, with many tests showing perfect accuracy. The results of the experiments highlight the random forest's strength in classification tasks on categorical data and the SVM's competence on continuous attributes.

1 Introduction

In the field of supervised machine learning, a plethora of models have been developed to find an input-output mapping function based on labeled training examples. Two of these models, the support vector machine and random forest, have displayed powerful capabilities in classification problems with even simple implementations. In this project, we were tasked with constructing and testing the algorithms on several datasets in both artificial and real-world domains. This work aims to identify the relative strengths and weaknesses of the models in learning tasks.

Ensemble methods are a statistical technique where many simple learning algorithms are combined together to make predictions. Although the individual models do not generate accurate results on their own, aggregating the predictions of dozens or hundreds of these weak learners can produce highly accurate outcomes. Random forests use a concept known as bootstrap aggregation to create the individual models utilized for predictions. Ensembling is primarily used as a variance-reduction measure in machine learning. Random forests are built on the decision tree algorithm, a high-variance, low-bias model, making it a perfect candidate for ensembling.

The support vector machine algorithm is constructed around the idea of maximizing the separation, or margin, between classes in a dataset. Finding the maximum gap allows the algorithm to fit a decision boundary more resistant to variance than other hyperplane-finding algorithms such as logistic regression. The real strength of the SVM is displayed when using kernels, a way of projecting data into higher-dimension feature spaces. Utilizing kernels, SVM's can find complex decision boundaries which would not be observable in non-linear and lower dimension spaces.

2 Data

The random forest and SVM models were tested on several datasets to evaluate their performance in supervised learning tasks. For this project, six datasets were utilized: two 'artificial' sets composed of computer-generated data and four real-world sets collected from UC-Irvine's machine learning repository. The differences between the data allow us to evaluate what contexts the models perform best in, informing future decision-making of when to apply the learning models constructed in this project.

2.1 Artificial Domains

The artificial domains consist of two datasets generated by a programming script. The spirals data contains 1,000 points and is separated into two classes, where each spiral belongs to a separate label. The spirals data presents a difficult classification task and is a good test for evaluating the model's performance in non-linear decision spaces. On the other hand, the 'blobs' data is created by producing k Gaussian data sources in an n-dimensional space. For the purposes of this project, several values of k were chosen to evaluate the model's ability to categorize noisy data in high-dimensional spaces. figure 1 shows a scatterplot of both datasets.

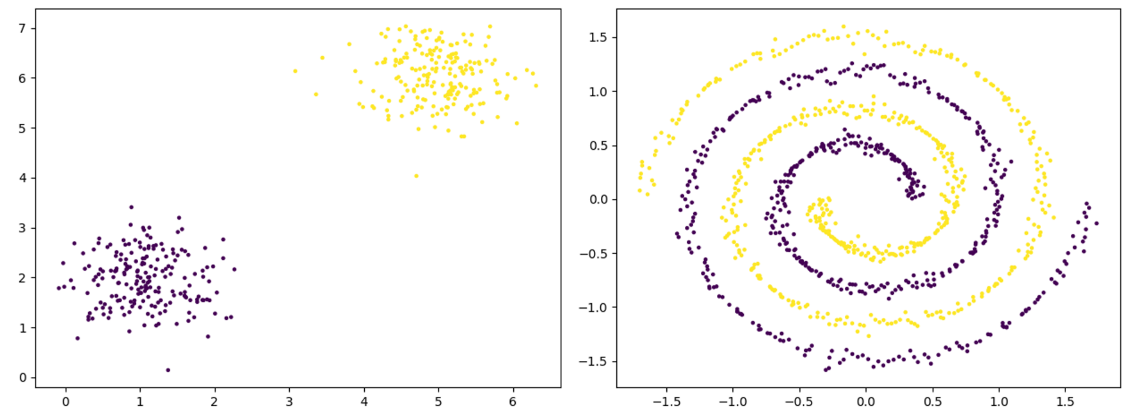
Figure 1: Graph of Blob and Spiral Data

2.2 MNIST Dataset

Similar to the first project, the models were tested on the Modified National Institute of Standards and Technology (MNIST) database, which contains a comprehensive collection of hand-written numbers (Deng, 2012). The data is represented by intensity values for each image pixel (Pixel0, Pixel1, ... , Pixel783). The original dataset contained continuous-valued numbers between 0 and 255, indicating the intensity value for each pixel. To ease computation requirements, intensity values were discretized into 5 bins and the model was tested on the modified data. The images in figure 2 depict examples of the hand-written digits included in the dataset.

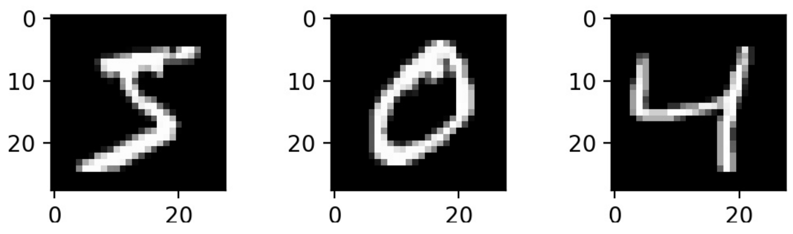
Figure 2: Examples of Digit Data

2.3 Adult Dataset

Utilizing information from the 1994 Census database, the adult dataset contains records of over 45,000 adults in the United States (Kohavi et al., 1996). A broad range of demographic statistics are included for each record in the data, including educational level, occupation, age, sex, race, among several others. These records are intended to predict whether an individual will earn above $50,000 a year in total pay. The adult dataset includes many continuous-valued and categorical features, making it very informative for evaluating model performance in a variety of data types.

2.4 Mushrooms Dataset

The choices for the final two datasets were left to the discretion of each team. Our first choice is a dataset containing samples of 23 species of mushrooms in the Agaricus and Lepiota families, used to determine if a mushroom sample is edible or poisonous (Schlimmer, 1987). Mushroom records were sampled from the Audobon Society Field Guide to North American Mushrooms and contains over 20 categorical features for each record. This dataset was chosen explicitly due to the lack of continuous-valued attributes, allowing us to evaluate model performance solely on categorical features.

2.5 Car Evaluation Dataset

The final dataset in this project is used to evaluate cars according to a collection of sale information (Bohanec & Rajkovic, 1988). The features include buying/maintenance price, capacity, and other technical characteristics utilized to determine the appeal of each car. This dataset was derived from a hierarchal decision model and has proven helpful in constructive induction methods. For the purposes of model evaluation, this data is useful as it contains mostly categorical features along with a few real-valued attributes.

3 Random Forest Model

3.1 Decision Trees

Decision trees are a visual and explicit representation of decisions and decision-making. This form of supervised learning uses a tree-like model and covers both classification and regression fields, but for the sake of this project, the focus will be on classification. A Decision tree is formed with its root at the top where each node represents a condition, and each leaf for that condition represents an outcome. The image in figure 3 depicts the structure of a decision tree for a classification dataset. The distinction between classification and regression trees is that the classification tree is meant to determine the state of an item based on a given set of features, and regression trees are meant to determine continuous values of an item.

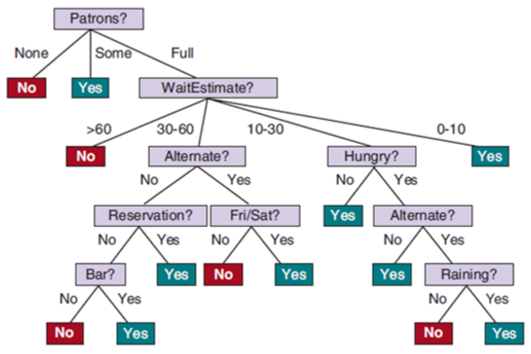

Figure 3: Decision Tree Structure

At each node in the tree, a choice must be made on which attribute to split on. This decision is defined by a cost function, which uses information about possible data splits to determine the optimal choice. The split method utilizes a recursive greedy partitioning algorithm to maximize the decrease in loss, with the goal of determining the most homogeneous branches in the data. Given a region R and loss L, we wish to choose a split which maximizes the decrease in loss. This can be formulated as:

$$L(R_p) - \frac{|R_1|L(R_1) + |R_2|L(R_2)}{|R_1| + |R_2|}$$

Where $L(R_p)$ refers to the loss of the parent, and the fraction subtracted represents the cardinality-weighted loss of the two children, R_1/R_2. There are multiple methods of defining the loss function for this equation, three of which were utilized in this project. The first loss function $L_{misclass}$ is known as misclassification loss and is understood as the number of examples that would be incorrectly classified under the chosen split:

$$L_{misclass}(R) = 1 - \max_c(\hat{p_c})$$

Although the misclassification loss function makes intuitive sense, it is not the best choice for split methods. This loss function does not guarantee that child nodes will make fewer errors than the parent, an undesirable outcome when attempting to minimize the rate of incorrect classification. Accordingly, two additional loss functions are developed to eliminate this possibility. The first is known as cross-entropy, a function from information theory used to measure the number of bits needed to represent an event. The loss function is defined as:

$$L_{cross-entropy}(R) = -\sum_c \hat{p_c} log_2 \hat{p_c}$$

The principal benefit of this loss function is its convex nature. When computing the loss of the two child regions, cross-entropy guarantees that the children will make fewer errors than the parent node. This feature of cross-entropy loss is why it is regarded as superior compared to misclassification. The final loss function is the Gini index, another convex function that guarantees a decrease in loss from parents to child nodes.

$$L_{gini}(R) = -\sum_c \hat{p_c}(1 - \hat{p_c})$$

Applying the loss functions to the parent and child nodes produces 'information gain,' which is the measure used to pick a split feature. For categorical data, information gain is computed for each feature, and the maximum value is chosen. For continuous-valued data, we created a set of all values and computed the information gain of splitting on each number. Once again, the number split which produces the highest information gain is chosen for that feature. Tests were performed using all three split methods to determine which performs best on the various datasets.

A final note on decision trees relates to hyperparameters. As decision trees can perfectly represent a training set with enough nodes, overfitting is common. To prevent this, limiting both the maximum depth of a decision tree and the minimum number of samples required to split on a feature is customary. This way, the decision tree cannot create leaf nodes for small subsets of the data not likely to be present in the test set.

3.2 Random Forest Ensembling

Random forests are a form of ensemble learning that combines many decision trees' outcomes to generate predictions. Random forests employ a method known as bootstrap aggregation, or bagging, to reduce the variance of the model. If we have a true population P that we wish to create a model for, a training set S is chosen from the population, such that $S \sim P$. Making the assumption that $S = P$ (although this is not necessarily true) allows the creation of a new bootstrap set Z which is generated by sampling with replacement from S, such that $Z \sim S$ and $|Z| = |S|$. This process is repeated M times, bringing about many bootstrap samples $Z_1, Z_2, ..., Z_M$, otherwise known as forest size.

In ensemble learning, a model (G_m) is trained on each bootstrap sample. When making predictions, each model is given the test set, and the majority vote of all models is taken. This is known as an aggregate predictor and is formalized as:

$$G(X) = \sum_{c} \frac{G_m(X)}{M}$$

The element of random forests which makes them 'random', relates to which features each model is trained on. For each node of a decision tree, a random subset of all features size n is chosen, and the tree is split on one of those features. This helps in reducing the correlation between the models and produces an overall decrease in variance. In general, it is recommended that the feature subset size is equal to the square root of the number of features. In this project, multiple feature subset sizes were tested as hyperparameters during cross-validation.

4 SVM Model

The fundamental motivation behind Support Vector Machines is finding a decision boundary that maximizes the separation, or margin, between classes in a dataset. Although this goal is only achievable in linearly separable data, the notion of using margin between points to fit a decision line has proven highsly effective in classification problems. The use of maximum-margin hyperplanes combined with the Kernel trick gives SVM models powerful learning capabilities in high-dimensional, non-linear feature spaces.

4.1 Margins

Margins allow us to incorporate 'confidence' into predictions when training a learning model. The further a point is away from a decision boundary, the more confident we are in a prediction. On the other hand, points very close to a separating hyperplane are more likely to be incorrectly classified with variation in the boundary. Because of this notion, the SVM algorithm attempts to fit a boundary which maximizes the 'confidence' for the most points in the data. In other words, SVM's maximize the margin between the decision boundary and the closest points.

The first step in formalizing the concept of margins is introducing a new classification notation for the SVM algorithm. Instead of the traditional class labels of $\{0, 1\}$, the labels y are represented as $y \in \{-1, 1\}$. Additionally, instead of parameterizing the linear classifier with the θ vector, the classifier is written as:

$$h_{w,b}(x) = g(w^T x + b)$$

Where parameters w, b are the slope and intercept of the decision line. In this formalization, $g(z) = 1$ if $z \geq 0$ and $g(z) = -1$, otherwise. This equation allows us to develop two versions of margin used to create the SVM algorithm. The first is known as functional margin, understood as the largest distance to the nearest training point from the separating hyperplane. For a training example $(x^{(i)}, y^{(i)})$ the functional margin is defined as:

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x + b)$$

An issue with using this equation for margin calculations regards scaling the w and b parameters. Scaling both w and b by a factor of 2 would result in the same functional margin as the original equation ($g(w^T x + b) = g(2w^T x + 2b)$). Due to this problem, it is customary to impose a normalizing factor on the parameters such as $||w||_2 = 1$, resulting in (w, b) being replaced with $(w/||w||_2, b/||w||_2)$. Finally, because only the points closest to the decision boundary affect calculations for the hyperplane, the functional margin for a training set can therefore be expressed as:

$$\hat{\gamma} = \min_{i=1,\ldots,m} \hat{\gamma}^{(i)}$$

The second version of margin in SVM's is known as the geometric margin ($\gamma^{(i)}$). Consider a point A in a training dataset. The distance between A and the closest point on the decision boundary B is given by $\gamma^{(i)}$. To find the value of $\gamma^{(i)}$, we first calculate $w/||w||$, a unit-length vector pointing in the direction of the w parameter. Accordingly, B is expressed as $x^{(i)} - \gamma^{(i)} \frac{w}{||w||}$. Since B lies on the decision boundary, $w^T x + b = 0$ at this point, thus:

$$w^T \left(x^{(i)} - \gamma^{(i)} \frac{w}{||w||} \right) + b = 0$$

Solving for $\gamma^{(i)}$, and incorporating $y^{(i)}$ into the equation, results in:

$$\gamma^{(i)} = y^{(i)} \left(\left(\frac{w}{||w||} \right)^T x^{(i)} + \frac{b}{||w||} \right)$$

This formalization of margin is preferable to functional margin because the resultant value is invariant to scaling of the w and b parameters. As discussed above, if we impose the constraint that $||w|| = 1$, then the functional margin and geometric margin are the same. Similar to the equation for functional margin, geometric margin is calculated for the closest point to the decision boundary, expressed as:

$$\gamma = \min_{i=1,\ldots,m} \gamma^{(i)}$$

4.2 Optimal Margin Classifier

The next logical step in developing the SVM is to derive an optimization problem which maximizes the geometric margin in a training set. Solving this equation allows the creation of decision boundaries with the best possible fit to the data, separating the classes with the largest possible gap. For the initial derivation of this problem, we assume that the training set is linearly separable between classes. An initial representation of this is given by:

$$\begin{aligned} \max_{\gamma,w,b} \;& \gamma \\ \text{s.t.} \;& y^{(i)}(w^T x^{(i)} + b) \geq \gamma, \;\; i = 1, ..., m \\ & ||w|| = 1 \end{aligned}$$

Although this equation gives the correct structure for the optimization, it is intractable to solve due to the non-convex $||w|| = 1$ constraint. To be solved by optimization software, the objective function must be converted into a convex form. This can be accomplished by incorporating the idea of scaling factors from the functional margin derivation discussed above. A scaling constraint is added to the functional margin of w, b such that:

$$\hat{\gamma} = 1$$

Next, because the geometric margin is simply the functional margin divided by $||w||$ if $||w|| = 1$, the geometric margin can be replaced in the optimization equation by $\frac{\hat{\gamma}}{||w||}$. A final change is recognizing that maximizing $\hat{\gamma}/||w|| = 1/||w||$ is equivalent to minimizing $||w||^2$. Thus, by plugging in the scaling constraint into the optimization problem, we achieve:

$$\begin{aligned} \min_{\gamma,w,b} \;& \frac{1}{2}||w||^2 \\ \text{s.t.} \;& y^{(i)}(w^T x^{(i)} + b) \geq 1, \;\; i = 1, ..., m \end{aligned}$$

While it is possible to solve the posed optimization problem, there is a change in the formulation which makes the algorithm easier to develop and much more powerful. Using the method of Lagrangian multipliers, an alternate 'dual-form' of the equation can be obtained that allows us to solve the primal problem outlined above and incorporate the Kernel trick. This dual form is acquired by getting the Lagrangian of the optimization problem and solving for the primal variables that minimize the objective function. The Lagrangian formulation of the problem is expressed as:

$$\mathcal{L}(w, b, \alpha) \;\; \frac{1}{2}w^T w - \sum_{i=1}^{m} \alpha_i [y^{(i)}(w^T x^{(i)} - b) - 1]$$

In this form, α is the Lagrangian multipliers vector, containing a value for each point in the training set. To derive the dual form, the Lagrangian $\mathcal{L}(w, b, \alpha)$ should be minimized with respect to w and b. By taking the partial derivatives of the lagrangian with respect to these parameters and setting them equal to zero, we obtain:

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$$

and

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

Plugging the definition of w into the original Lagrangian and adding the required constraints, we get:

$$\begin{aligned}\max_{\alpha} \;\; W(\alpha) = & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \;\; & \alpha_i \geq 0, i = 1, ..., m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0\end{aligned}$$

Because there are only a few points in the training set which will act as support vectors for the decision boundary, the majority of points in α_i will be zero. This greatly simplifies the calculation of the dual equation since most α values will not contribute to the sum. After the maximum α's are obtained from the optimization problem, it is easy to take those values and solve for the optimal w^*, along with b^* utilizing:

$$b^* = -\frac{\max_{i:y^{(i)}=-1} w^{*T} x^{(i)} + \min_{i:y^{(i)}=1} w^{*T} x^{(i)}}{2}$$

4.3 The Kernel Trick

The effort put into converting the original optimization problem to its Lagrangian dual form did have a purpose, as it allows us to incorporate the Kernel trick into the SVM. By transforming the optimization equation to be in terms of the inner product $\langle x^{(i)}, x^{(j)} \rangle$, we can easily substitute in a Kernel equation, allowing the SVM to view the dataset in high-dimensional, non-linear decision spaces.

The traditional approach to projecting data into higher-dimensional feature spaces is by using some mapping equation $\phi(x)$. For example, if we wish to take some feature and represent it as a squared and cubic version, the following feature mapping equation is utilized:

$$\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}$$

To incorporate this mapping into the SVM, $\phi(x)$ replaces x in all locations. As the algorithm is written in terms of the inner product $\langle x^{(i)}, x^{(j)} \rangle$ ($\langle x, z \rangle$ for simplicity), those inner products would therefore be replaced by $\langle \phi(x), \phi(z) \rangle$. The corresponding Kernel equation is understood as:

$$K(x, z) = \phi(x)^T \phi(z)$$

Given ϕ, it is relatively simple to compute the Kernel by calculating $\phi(x)$ and $\phi(z)$ and taking the inner product. But defining a smart Kernel equation can substantially simplify the number of steps needed to compute this. Consider the following Kernel function:

$$K(x, z) = (x^T z)^2$$

Applying this to the inner product achieves the same result as using the following mapping function to x and z and taking the inner product:

$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix}$$

The difference between the Kernel and mapping functions is that calculating the high-dimensional ϕ requires $O(n^2)$ time while $K(x, z)$ requires only $O(n)$, a significant reduction in complexity. Utilizing a variety of Kernel functions, SVM's can project data into high (or even infinitely large) feature spaces, while still maintaining a small runtime complexity.

4.4 Non-Separable Case

The assumption that a dataset is linearly separable almost never holds true in real-life cases. As a consequence, it is necessary to update the optimum margin classifier to incorporate non-separable cases. Utilizing the concept of $\ell1$ regularization, we can update the original optimization equation as follows:

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{m} \xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \;\; i = 1, ..., m \\ & \xi_i \geq 0, \;\; i = 1, ..., m \end{aligned}$$

By adding the desensitization term ξ, training examples can have a functional margin of less than 1. The C term is known as the cost, which adds a penalty to the optimization problem for examples with functional margin $= 1 - \xi_i$. Moreover, the Lagrangian form of the optimization equation can also be updated using this concept:

$$\mathcal{L}(w, b, \xi, \alpha, \beta) = \frac{1}{2}w^T w + C\sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i[y^{(i)}(w \cdot x^{(i)} + b) - 1 + \xi_i] - \sum_{i=1}^{m} \beta_i \xi_i$$

In this expression, α and β are the Lagrange multipliers, with β now added for the regularization term. Following similar steps as above to derive the dual form, the final optimization problem is obtained:

$$\begin{aligned}
\max_{\alpha}\ & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\
\text{s.t.}\ & 0 \le \alpha_i \le C,\ \ i = 1, ..., m \\
& \sum_{i=1}^{m} \alpha_i y^{(i)} = 0
\end{aligned}$$

4.5 SMO Algorithm

The final piece of the puzzle in creating the SVM is an algorithm used to solve the Lagrangian dual problem. The sequential minimal optimization (SMO) algorithm provides a simple mechanism for achieving this result. The full SMO algorithm is useful as it is guaranteed to converge to an optimal solution, utilizing a slew of optimization principles to efficiently generate results. For this paper, a simplified version of the SMO algorithm is used which does not guarantee convergence for all datasets.

SMO uses an optimization approach known as coordinate ascent to choose the best α_i's. This method works by selecting and varying a single α_i while holding the remaining values constant. By varying the selected α, coordinate ascent can iterate over many potential parameter values and select the optimal one for the output function.

When applying the coordinate ascent approach to the optimization problem, a small change must be made to fit the dual form. If only one α_i is chosen for coordinate ascent, no progress can be made due to the following constraint:

$$\alpha_1 y^{(1)} = -\sum_{i=2}^{m} \alpha_i y^{(i)}$$

This condition shows that α_1 is determined by the remaining, fixed α's, meaning that only adjusting one value does not produce the optimal solution. Accordingly, the simplifed SMO algorithm makes use of two α's for coordinate ascent, rather than just one. The updated method thus selects some pair α_i and α_j and reoptimizes $W(\alpha)$ with respect to the chosen values. Similar to the previous method, the remaining α's are fixed, and the algorithm repeats until convergence.

One of the most important aspects of the SMO algorithm is selecting which α values to optimize with. This is especially important for an efficient algorithm, as there are $m(m-1)$ potential choices for the α pair. In the simplified SMO, we iterate over all α_i's from $i = 1, \ldots, m$. If any α_i does not satisfy the KKT conditions for convergence, a random α_j is chosen for joint optimization. Since all possible α_i, α_j pairs are not optimized, this simplified algorithm is not guaranteed to converge.

Once the α pair is chosen, the next step is to compute the constraints on these values before solving the constrained optimization problem. To satisfy the constraint of $0 \leq \alpha_j \leq C$, we must find bounds, L and H, such that $L \leq \alpha_j \leq H$, given by the following:

$$\begin{aligned} &\text{If } \; y^{(i)} \neq y^{(j)}, \quad L = max(0, \alpha_j - \alpha_i), H = min(C, C + \alpha_j - \alpha_i) \\ &\text{If } \; y^{(i)} = y^{(j)}, \quad L = max(0, \alpha_i + \alpha_j - C), H = min(C, \alpha_i + \alpha_j) \end{aligned}$$

The optimal α_j is found with:

$$\alpha_j := \alpha_j - \frac{y^{(j)}(E_i - E_j)}{\eta}$$

where

$$\begin{aligned} E_k &= f(x^{(k)}) - y^{(k)} \\ \eta &= 2\langle x^{(i)}, x^{(j)}\rangle - \langle x^{(i)}, x^{(i)}\rangle - \langle x^{(j)}, x^{(j)}\rangle \end{aligned}$$

To convert calculations to a Kernelized form, simply replace the inner products within η with the Kernel function. It is possible that the resultant α_j value will fall outside of the L and H boundaries established for the parameter. In this case, we simply truncate the value according to the following:

$$\alpha_j := \begin{cases} H & \text{if } \; \alpha_j > H \\ \alpha_j & \text{if } \; L \leq \alpha_j \leq H \\ L & \text{if } \; \alpha_j < L \end{cases}$$

We can use the calculated values to compute α_i:

$$\alpha_i := \alpha_i + y^{(i)} y^{(j)} (\alpha_j^{(old)} - \alpha_j)$$

After optimizing α_i and α_j, the final step in SMO is to compute the b threshold, given by:

$$b := \begin{cases} b_1 & \text{if } \; 0 < \alpha_i < C \\ b_2 & \text{if } \; 0 < \alpha_j < C \\ (b_1 + b_2)/2 & \text{otherwise} \end{cases}$$

where

$$\begin{aligned} b_1 = b - E_i - y^{(i)}(\alpha_i - \alpha_i^{(old)})\langle x^{(i)}, x^{(i)}\rangle - y^{(j)}(\alpha_j - \alpha_j^{(old)})\langle x^{(i)}, x^{(j)}\rangle \\ b_2 = b - E_j - y^{(i)}(\alpha_i - \alpha_i^{(old)})\langle x^{(i)}, x^{(j)}\rangle - y^{(j)}(\alpha_j - \alpha_j^{(old)})\langle x^{(j)}, x^{(j)}\rangle \end{aligned}$$

Note that if both $0 < \alpha_i < C$ and $0 < \alpha_j < C$ are true, than either b threshold will work.

5 Random Forest Experimental Results

5.1 Artificial Domains

The first tests were performed on the artificial datasets, including the Gaussian blobs and spirals program-generated data. For the spirals data, all three split methods were tested on forest sizes of 50, 100, and 150 trees. The results of the experiment are depicted in table 1. The random forest spirals test produced an interesting result counter-intuitive to traditional thinking on decision trees. The cross-entropy and GINI split methods usually generate better feature splits due to the convex nature of the functions. In this experiment, the misclassification split method produced the best model accuracy, classifying 90% of points correctly with a forest size of 150.

	Forest Size		
Split Method	50	100	150
Cross-Entropy	64.0	61.0	66.0
GINI Index	69.0	66.5	75.0
Misclassification	84.0	83.0	90.0

Table 1: Random Forest Model Accuracy on Spirals Dataset

For the blobs dataset, several tests were performed with varying 'k' values, which indicate the number of Gaussian sources present in the training/test sets. For $k = 2, 3$, the model had no issues accurately categorizing the blobs, with over 99% accuracy in the majority of tests. Due to limited computation resources and time, the tests with $k = 4, 5$ were performed with forest sizes of 25, 50, and 75. Despite the smaller forest sizes, the model performed very well in classifying the data. All tests achieved a model accuracy of over 98%, with two classifying all test examples correctly.

	2 Gaussians			3 Gaussians		
	Forest Size			Forest Size		
Split Method	50	100	150	50	100	150
Cross-Entropy	100	100	100	98.89	99.44	100
GINI Index	98.75	100	100	99.44	100	98.89
Misclassification	98.75	100	100	99.44	99.44	99.44

Table 2: Random Forest Model Accuracy on Blobs Dataset (k=2, 3)

	4 Gaussians			5 Gaussians		
	Forest Size			Forest Size		
Split Method	25	50	75	25	50	75
Cross-Entropy	100	99.58	98.75	100	98.0	98.0
GINI Index	98.75	99.17	99.58	99.33	99.0	98.67
Misclassification	99.17	99.58	99.58	98.33	99.0	98.67

Table 3: Random Forest Model Accuracy on Blobs Dataset (k=4, 5)

5.2 MNIST Dataset

The random forest model performed similarly to project 1's Naïve Bayes model on the MNIST database. The hand-written digits data contained by far the most features of all the datasets, making it an optimal test for the feature subset hyperparameter. As expected, the model performed poorly when the feature subset was small, and accuracy quickly grew with a larger subset. This intuitively makes sense for the MNIST data, as models considering only 5 of over 700 pixels cannot generate consistently accurate predictions.

	Forest Size								
	25			50			75		
Split	Feature Subset Size			Feature Subset Size			Feature Subset Size		
Method	5	10	15	5	10	15	5	10	15
Cross-Entropy	67.26	73.25	76.86	69.57	75.44	81.19	72.25	85.88	86.31
GINI Index	68.87	73.26	81.11	69.17	75.54	79.16	71.77	83.99	82.67
Misclassification	69.97	74.58	78.33	73.52	75.68	80.97	76.57	78.11	84.02

Table 4: Random Forest Model Accuracy on MNIST Dataset

The results of the MNIST experiment are shown in table 4. For cross-entropy and GINI split methods in larger forest and feature subset sizes, the model correctly classified around 83-86% of digits, slightly above the mark of Naïve Bayes. This experiment would benefit a lot from a longer test period, as the forest sizes were limited to 25, 50, and 75 due to the large size of the dataset. To more accurately evaluate the model's performance on the digit data, we would prefer to test more feature subset sizes on a larger forest.

5.3 Adult Dataset

The smallest variation in experimental results was obtained during the adult dataset test. Despite varying the split method, feature subsets, and forest sizes, no discernible improvement in model accuracy was shown. The random forest predicted the income of around 83-84% of adults correctly in all tests. Table 5 shows the experimental results for the adult data. Compared to the various models outlined in Platt's paper, the random forest implementation produced very similar results.

	Forest Size					
	25		50		100	
Split	Feature Subset Size		Feature Subset Size		Feature Subset Size	
Method	2	4	2	4	2	4
Cross-Entropy	83.12	83.37	83.99	84.86	83.95	85.74
GINI Index	84.26	84.84	83.87	85.77	84.28	85.34
Misclassification	82.63	84.65	83.11	83.37	83.83	83.22

Table 5: Random Forest Model Accuracy on Adult Dataset

5.4 Mushrooms Dataset

The random forest model truly shined when tested on the Mushrooms dataset. The model classified 100% of data instances correctly in 20/27 of the tests. As the number of trees in the forest grew, the results quickly converged to perfect accuracy. At a forest size of 50, the cross-entropy method produced slightly better results but that performance-gap closed once the number of decision trees grew.

	Forest Size								
	50			100			150		
Split	Feature Subset Size			Feature Subset Size			Feature Subset Size		
Method	2	4	6	2	4	6	2	4	6
Cross-Entropy	100	99.39	100	100	100	100	100	100	100
GINI Index	99.42	100	98.58	100	100	100	100	100	100
Misclassification	99.66	98.32	98.81	100	100	99.11	100	100	100

Table 6: Random Forest Model Accuracy on Mushrooms Dataset

5.5 Car Evaluation Dataset

In the final test, the random forest model was evaluated on the car dataset. At its best, the model classified 96% of test instances correctly, but the majority of results hovered around the 88% to 92% mark. Although the results slightly improved as the feature subset size and forest grew, the change was marginal at best. This dataset is an excellent example of the limits of hyperparameter selection for the random forest model in some cases. Although the model performed well in the tests, fine-tuning the forest and feature subset sizes did not generate more accurate results as seen in other experiments.

	Forest Size								
	50			100			150		
Split	Feature Subset Size			Feature Subset Size			Feature Subset Size		
Method	2	4	6	2	4	6	2	4	6
Cross-Entropy	89.60	94.22	91.62	90.75	92.49	90.17	92.49	91.33	90.17
GINI Index	88.73	92.77	93.64	92.20	92.20	91.91	89.88	96.24	90.17
Misclassification	87.28	92.19	90.46	91.33	93.06	93.93	86.13	92.49	91.04

Table 7: Random Forest Model Accuracy on Car Evaluation Dataset

6 SVM Experimental Results

6.1 Artificial Domains

Similar to the random forest, the first experimental tests were conducted on the artificial datasets. For cross-validation purposes, the inner-product (basic SVM), Gaussian kernel, and polynomial $K(x,z) = (x^T z + c)^2$ kernel were each tested for C regularization parameter values of 1, 50, 100, and 1000. The results for the spirals dataset is presented in table 8.

	C Parameter			
Kernel	1	50	100	1000
Inner-Product	49.85	49.35	49.65	50.25
Polynomial	45.55	46.50	46.0	45.80
Gaussian	59.20	69.80	83.15	96.80

Table 8: SVM Model Accuracy on Spirals Dataset

The basic inner-product and polynomial kernels did not result in high model accuracy, with the SVM classifying less than half of examples correctly during the tests. On the other hand, the Gaussian kernel combined with a high C regularization value produced very accurate results. The best model utilized a Gaussian kernel with a C of 1000 and achieved almost 97% accuracy.

For the Gaussian blobs data, the SVM was tested on several sets with varying 'k' values. The results of these tests are depicted in tables 9 and 10. In the tests with $k = \{2, 3, 4, 5\}$, the inner-product method exhibited the strongest performance with model accuracy of over 99% in all tests. The polynomial and Gaussian kernels performed at a similar level when the regularization parameter was high, but struggled when it was low. These results highlight how many SVM kernels can generate accurate results, but it is especially important to find the right hyper-parameter values with cross-validation.

	2 Gaussians				3 Gaussians			
	C Parameter				C Parameter			
Kernel	1	50	100	1000	1	50	100	1000
Inner-Product	100	100	100	100	99.94	100	99.89	99.89
Polynomial	100	100	100	100	32.56	81.50	87.67	92.89
Gaussian	100	100	100	100	33.44	98.0	100	99.78

Table 9: SVM Model Accuracy on Blobs Dataset (k=2, 3)

	4 Gaussians				5 Gaussians			
	C Parameter				C Parameter			
Kernel	1	50	100	1000	1	50	100	1000
Inner-Product	100	100	100	100	99.03	99.30	99.27	99.03
Polynomial	23.92	73.92	83.92	91.33	18.90	48.40	63.50	84.43
Gaussian	23.46	66.79	87.25	100	18.20	32.73	57.23	98.90

Table 10: SVM Model Accuracy on Blobs Dataset (k=4, 5)

6.2 MNIST Dataset

The SVM demonstrated its worst performance when tested on the MNIST database. All models tested could not classify more than 15% of test examples correctly, with multiple achieving accuracy below what is expected by random chance. This outcome is likely due to an error in handling the data in our code, but we could not locate the source of the issue after extensive debugging.

	C Parameter			
Kernel	1	50	100	1000
Inner-Product	10.26	12.26	11.81	12.30
Polynomial	11.64	11.04	10.93	11.56
Gaussian	10.25	9.94	9.73	10.06

Table 11: SVM Model Accuracy on MNIST Dataset

6.3 Adult Dataset

Following the lead of Platt's paper on the SMO algorithm, we tested the model on the adult dataset. In the original SMO paper, Platt tested the model on the adult data with a regularization parameter of 0.5, so we included that in the cross-validation hyperparameter list (Platt, 1998). Moreover, the categorical variables were discretized into bins to allow the model to fit to the data.

	C Parameter			
Kernel	0.5	1	50	100
Inner-Product	73.21	76.56	76.23	74.73
Polynomial	79.27	76.11	80.10	78.70
Gaussian	77.11	78.95	74.49	79.90

Table 12: SVM Model Accuracy on Adult Dataset

The results of the adult data experiment are shown in table 12. For this test, all hyperparameter combinations displayed roughly the same performance. The inner-product method resulted in a model accuracy score of around 75%, while the kernelized SVM methods produced a score of almost 80%, on average. These results are slightly below the average accuracy obtained by the random forest model.

6.4 Car Evaluation Dataset

The final test was performed on the car evaluation dataset. Similar to the adult data, the categorical columns were transformed into bins so the SVM could train. Compared to the random forest model, the SVM using polynomial and Gaussian kernels performed substantially worse. Across all regularization parameter values, the kerneled SVM models averaged only 70% accuracy in predicting car quality. The basic SVM using the inner-product method performed much better, achieving an accuracy rate of 91.33% with a C of 1000. Table 13 shows the results of the car evaluation tests.

	C Parameter			
Kernel	1	50	100	1000
Inner-Product	84.39	84.85	89.02	91.33
Polynomial	68.21	76.01	68.79	70.52
Gaussian	68.21	68.21	71.10	69.94

Table 13: SVM Model Accuracy on Car Evaluation Dataset

7 Discussion

7.1 Random Forest

Implementation of decision trees and the random forest were fairly straightforward. The most challenging aspect was defining a split-picking method that accounts for both categorical and continuous data. We found that separating the data by the attribute type was the best way to calculate the information gain for all features. In python, a dictionary was created which stored the information gain from each attribute, and the maximum was selected for splitting purposes. Furthermore, when making predictions on a test set, we created a simple recursive method to traverse the nodes of the tree and categorize each data instance.

As noted above in the experimental results section, there were a few surprising outcomes of the tests. Most notably, the misclassification loss split method performed best on the spirals dataset. The difference in model accuracy was quite significant, with models using misclassification averaging almost 20% higher. This is a significant result as it highlights that conventional thinking about loss functions does not always hold true in tests. It also shows the importance of cross-validation when training a model.

In general, the random forest model exhibited the best performance on datasets with more categorical features. The mushrooms and car evaluation datasets contain mainly categorical features, and the model performed best during those tests. Moreover, the model tended to achieve better results as the forest and feature subset sizes grew. This makes intuitive sense, as increasing the number of trees in the forest helps drive down variance, and considering more features lets the trees garner a more comprehensive view of the data when deciding splits. Aside from the outlier case of the spirals data, the convex loss functions of cross-entropy and Gini tended to perform best.

In future experiments, we would like to repeat the tests more and use an expanded set of hyperparameters to examine what improves/reduces the accuracy of the random forest. Although the algorithms implemented seemed to function correctly in the tests, it took a significant amount of time to train the random forests, which reduced our ability to perform a wide breadth of tests. Specifically, choosing splits for continuous-valued features required a lot of computations when evaluating information gain for all possible split values. With more time, we would like to analyze all methods within the programs to determine what can reduce long training times.

7.2 SVM

The SMO algorithm was the most difficult model to implement of all the projects so far. Compared to the Naïve Bayes and random forest models, the SVM required significantly more mathematical equations for correct functionality. While previous projects required more in-depth comprehension of data structures, the SVM necessitated an understanding of python math and optimization packages. In the end it was not too difficult to get the equations working properly, but there was a learning curve to get down the correct syntax.

The SVM algorithm showed the best performance on continuous-valued datasets. On tests where categorical columns were converted into bins, the SVM did not display as strong of classification accuracy. Yet, in most tests there was one combination of kernel method and regularization parameter that produced highly accurate results. This shows the robustness of the SVM model, as carefully selecting different kernel functions and hyperparameter settings can produce a model capable of classifying many datasets accurately.

8 Related Work

8.1 Random Forest

Random forests are applicable to a wide array of problem sets. A common area where random forests have been applied in academic research regards human detection in images. In a study published by Keimyung University in Korea, researchers utilized a random forest model to locate the presence of humans with the INRIA dataset (D.-Y. Kim, Kwak, Ko, & Nam, 2012). Compared to other learning models, the random forest displayed the best performance in this task. In a similar vein of study, researchers at Jiangnan University used online random forests to create a facetracking algorithm, which achieved high accuracy in a very efficient manner (Bao & Zhang, 2015).

Extending beyond the challenge of locating people in images, random forests have also shown promise in analyzing the features of human faces. Age estimation has presented one of the toughest challenges for learning models to tackle, due to the vast differences in appearance across demographics. In a paper published in the IEEE Transactions on Pattern Analysis and Machine Intelligence, the authors combined convolutional neural networks with random forests to approach the task (Shen et al., 2019). This unique model demonstrated top-of-class performance in age estimation.

A final area where the random forest has exhibited success is in cybersecurity. Specifically, researchers in Finland applied random forests to help identify faults in IP networks which assists in intrusion monitoring (Seppälä et al., 2010). Machine learning already has an enormous role in cybersecurity operations, and optimizing model performance in this field performs an invaluable role in protecting private information online. Based on the implementations observed, there are many other problems that the Random Forest could be applied to successfully. It is important to limit the intricacies of the problem; otherwise, if there are too many features to take into account, the accuracy of the model will drastically change.

8.2 SVM

Although a simple version of the SVM was implemented in this project, an immense amount of academic research has been dedicated to improving the model's predictive capabilities through modifications to the algorithm. At the Pohang University of Science and Technology in South Korea, researchers developed an ensembling method for SVM's, using both bagging and boosting techniques (H.-C. Kim, Pang, Je, Kim, & Bang, 2003). When testing the ensemble methods on fraud detection, MNIST, and iris datasets, the model outperformed the traditional SVM implementation in all cases.

Separate research has shown promise in combining the SVM model with other supervised learning techniques. A study published at the Rensselaer Polytechnic Institute in New York implemented the traditional decision tree learning model, but with SVM's at each decision point (Bennett & Blue, 1998). The central goal of this research is to create a logically simple model capable of viewing data in high-dimensional spaces. By integrating SVM kernels with the simple design of decision trees, the merged models achieved great classification results while being able to generalize well.

Aside from attempting to modify the structure of the SVM algorithm, other papers have investigated practical applications of the supervised learning model. In the field of machine condition monitoring and fault diagnosis, researchers at Pukyong National University were one of the first to apply SVM to this learning problem (Widodo & Yang, 2007). This use for the SVM shows excellent promise, potentially helping lower maintenance costs and substantially improving productivity for machine systems.

In the realm of quantitative finance, an extensive amount of effort has been devoted to predicting market trends with advanced machine learning models. Accurate prediction of financial trends presents an invaluable economic opportunity, allowing investors to capitalize on small changes in stock activity. At the University of Oklahoma, experiments were conducted with the SVM model, attempting to forecast changes in several stock prices (Trafalis & Ince, 2000). The SVM tested in this paper performed similarly well to other advanced models, including backpropagation and RBF neural networks.

9 Conclusions

Both the random forest and SVM models showed strong performance as classifiers in different settings. While the random forest conveyed its strength on categorical data, the SVM performed best on continuous-valued attributes. The experimental results highlight the differing strengths of the models, and how smart applications of them can produce beneficial results in research. The unique statistical methods of ensembling and kernel functions helped transform relatively simplistic algorithms into highly capable learning models which can analyze data in complex feature spaces.

References

Bao, F., & Zhang, Y. (2015). Face tracking algorithm based on online random forests detection. In *2015 14th international symposium on distributed computing and applications for business engineering and science (dcabes)* (pp. 320–323).

Bennett, K. P., & Blue, J. (1998). A support vector machine approach to decision trees. In *1998 ieee international joint conference on neural networks proceedings. ieee world congress on computational intelligence (cat. no. 98ch36227)* (Vol. 3, pp. 2396–2401).

Bohanec, M., & Rajkovic, V. (1988). Knowledge acquisition and explanation for multi-attribute decision making. In *8th intl workshop on expert systems and their applications* (pp. 59–78).

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, *29*(6), 141–142.

Kim, D.-Y., Kwak, J.-Y., Ko, B. C., & Nam, J.-Y. (2012). Human detection using wavelet-based cs-lbp and a cascade of random forests. In *2012 ieee international conference on multimedia and expo* (pp. 362–367).

Kim, H.-C., Pang, S., Je, H.-M., Kim, D., & Bang, S. Y. (2003). Constructing support vector machine ensemble. *Pattern recognition*, *36*(12), 2757–2767.

Kohavi, R., et al. (1996). Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Kdd* (Vol. 96, pp. 202–207).

Platt, J. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines.

Schlimmer, J. C. (1987). *Concept acquisition through representational adjustment* (Unpublished doctoral dissertation). University of California, Irvine.

Seppälä, T., Alapaholuoma, T., Knuuti, O., Ylinen, J., Loula, P., & Hätönen, K. (2010). Implicit malpractice and suspicious traffic detection in large scale ip networks. In *2010 fifth international conference on internet monitoring and protection* (pp. 135–140).

Shen, W., Guo, Y., Wang, Y., Zhao, K., Wang, B., & Yuille, A. (2019). Deep differentiable random forests for age estimation. *IEEE transactions on pattern analysis and machine intelligence*, *43*(2), 404–419.

Trafalis, T. B., & Ince, H. (2000). Support vector machine for regression and applications to financial forecasting. In *Proceedings of the ieee-inns-enns international joint conference on neural networks. ijcnn 2000. neural computing: New challenges and perspectives for the new millennium* (Vol. 6, pp. 348–353).

Widodo, A., & Yang, B.-S. (2007). Support vector machine in machine condition monitoring and fault diagnosis. *Mechanical systems and signal processing*, *21*(6), 2560–2574.