An Exploration and Implementation of Machine Learning Algorithms

Quinlan Bock
Mathematics Department

Lawrence University

Abstract

The goals of this research project are to explore and understand the mathematics behind machine learning algorithms as well as the implementation of these algorithms in python. In addition, assess the algorithm performance on different data sets, the negatives and benefits that come with using any particular algorithm, as well as an analysis of the parameters that are used in the algorithms to create these models.

March 24, 2020

1 Research Questions

1.1 What are the negative and positive aspects of each method?

The goal of this project is to learn about various machine learning algorithms (mostly classification algorithms as opposed to regression) and get up and close with their inner workings and mathematics by writing python implementations of each of the algorithms. Through this process I will obtain a deeper understanding of these methods and will ultimately be able to analyze which method performs best given different parameters and look at the interpretablity of each algorithm as well as the computational resources the algorithm demands.

1.2 What method performs the best on each data set and why?

Looking at the accuracy of the predictions will allow me to examine and access the performance of each algorithm. Different data sets might each provide their own different and more challenging problems for the algorithms to model. The performance and knowledge of the mechanics of the algorithm might provide insight to why certain algorithms are better suited to model specific problems. Each of the models will be fitted on multiple data sets chosen from the University of California Irvine Machine Learning Repository multiple times, each time with different tuning parameters to assess each of their performances as well as gain insight into how the parameters affect the performance.

2 Significance

2.1 Data sets

Data sets come in various shapes and forms, ranging from small and relatively simple problems to large complicated ones. There is an abundance of many free to use, well curated data sets, so there is more than likely a good amount that will suit any given machine learning algorithm. These data sets can then be used to test how a model is performing. It is also likely that there are many well documented papers and bench marks using the same data sets that are applicable to that algorithm, so that its easy to know where your algorithm/model stands in relation to many others that have been developed. There are many data sets exist for the sole purpose of providing benchmarks to test models on, however, there are also data sets that stem from real problems where there is still on going research into how to better improve the predictions for them. A greater accuracy in these predictions can mean detecting tumors more reliably or making predictions in stock prices that can fuel investments. Whether data sets exist for the purpose testing models or for benefiting from new and better statistical models both help to progress the field of machine learning and aid those getting into the field.

2.2 Methods

Machine learning methods also bear great importance however complex or useful they might be. In the case of a cutting edge machine learning algorithm that has state of the art results on certain data sets it obviously has great use in the context of being used as tools of prediction for those problems and as inspiration to how to approach similar and other problems and data sets. Even simpler machine learning algorithms bear great significance. Even if their results don't nearly hold up to those of newer and more sophisticated algorithms they still serve as the basis for how many of these more sophisticated algorithms work. Understanding more complicated machine learning methods require a great understanding and mastery of the simpler ones.

2.3 Data sets

2.3.1 Wine Quality

The one data set that I will be using for regression is a data set on wine quality. There are 11 different chemical attributes about wine consisting of: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, and alcohol. These 11 attributes make up the input variable $x_i = (x_{i1}, x_{i2}, \ldots, x_{i11})^T$ and the response variable is the quality of wine on a scale of 1 to 10. Although the response data in the data set is finite we can treat it as a regression problem since any rating between two integer values has the real interpretation of being somewhere between not good as the higher value and better than the lower value.

2.3.2 IRIS

The IRIS data set is perhaps the most famous data set in all of machine learning. This data set consists of just 4 attributes: sepal length in cm, sepal width in cm, petal length in cm, and petal width in cm and presents the classification problem of trying to predict the response variable is which specific type of IRIS a flower is whether that be Setosa, Veriscolour, or Verginica. This data set is know for its simplicity of only 4 attributes and the relative ease with which these 3 species can be distinguished based on the relevant characteristics.

2.3.3 Banknote Authentication

The Banknote Authentication data set consists of 4 metrics gathered from pictures of counterfeit and real banknotes: variance of Wavelet Transformed image, skewness of Wavelet Transformed image, Kurtosis of Wavelet Transformed image, and entropy of image. From these 4 attributes the goal is to train a model to correctly identify the class of the banknote, that is, whether it is counterfeit or whether it is real.

2.3.4 Breast Cancer

The Breast Cancer data set is a data set that consists of one response nine different predictors: age, menopause, tumor-size, inv-nodes, node-caps, deg-malig, breast, breast-quad, and irradiat. Given these input variables for a single data point the desired response variable to classify whether or not there is a recurrence of breast cancer. This data set has been used in a vast number of papers looking at performance of machine learning classifiers.

3 Algorithms

3.1 Introduction

Most machine learning algorithms aim to create models that can describe and or predict a response variable y based on one or more input variables $x = (x_1, x_2, ..., x_p)$. This is done by using a training set; a bunch of data points where both the response variable and the input variables are used to train or tune parameters in each model or algorithm so that when given a new input point is not in the train set it can accurately predict it. There are two types of machine learning algorithms. Algorithms used for classification use the response variable of a point describes which class that point belongs to, where as algorithms used for regression use the response variable to denote some continuous variable. The following sections describe mainly machine learning algorithms used for classification and how they are constructed and trained.

3.2 Notation

For consistency the same notation to denote input and response variables will be used for all algorithms. The i^{th} data point in the data set with p attributes is denoted

$$x_i = (x_1, x_2, ..., x_p)^T = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix}$$

and its respective response variable is denoted y_i. The matrix with n data points is then denoted by

$$\boldsymbol{X} = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_p^T \end{pmatrix}$$

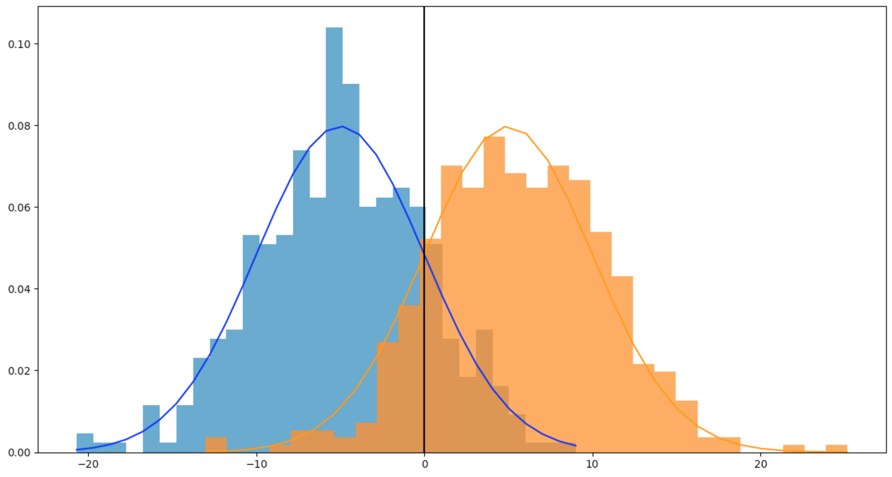
and finally the column vector denoting the j^{th} input variable of each the n data points is denoted

$$\boldsymbol{x}_j = \begin{pmatrix} x_{1j}^T \\ x_{2j}^T \\ \vdots \\ x_{nj}^T \end{pmatrix}$$

3.3 Linear Discriminant Analysis

The goal of Linear Discriminant Analysis (LDA) is to model the distribution of each of the p predictors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_p$ for each class k. Since we have modeled these distribution we have access to the probability that any given x_i takes on certain specific values of each of its parameters $x_i1, x_i2, ..., x_ip$ given that the class they belong to is k. However, what we want to find is the opposite of this and luckily we can flip the assumption of class k using Bayes Theorem to find the probability that a point belongs to class k assuming it's parameters $x_i1, x_i2, ..., x_ip$.

Figure 1: Two samples from a normal distributions both with $\sigma^2 = 5$ and $\mu_1 = -5$ and $\mu_2 = 5$



In the case that there is a single predictor and any number of classes the decision is easy to visualize, as seen in Figure 3 where there is one predictor and two classes it is clear that $P(\text{orange}|x > 0) > P(\text{blue}|x > 0)$ and $P(\text{blue}|x < 0) < P(\text{blue}|x < 0)$. The heuristic for LDA remains to model the distribution of X and use the distributions to find $f_k(x) = P(X = x|Y = k)$, that is the probability that the input variable X is approximately equal to x given that the response variable Y is equal to k, meaning it belongs to the k^{th} of which there are K of. Then continue by computing this statistic for each of the K separate classes of the response variable and classify x corresponding to the class that yields the highest $f_k(x)$. Bayes theorem and the law of total probability gives us

$$p_k(x) = P(Y = k|X = x) = \frac{P(X = x|Y = k)P(Y = y)}{\sum_{l=1}^{K} P(X = x|Y = l)P(Y = l)} \tag{1}$$

Then for the case that there is a single predictor and assuming normality we have that the probability density function (PDF) is given by

$$P(X = x|Y = k) = \frac{1}{\sqrt{2\pi}\sigma_k} e^{-\frac{1}{2\sigma_k}(x-\mu_k)^2} \tag{2}$$

where μ_k is a class specific mean vector and σ_k^2 is the variance associated with μ_k. Given this we can plug this into Equation 1 to get

$$p_k(x) = \frac{\frac{1}{\sqrt{2\pi}\sigma_k} e^{-\frac{1}{2\sigma_k}(x-\mu_k)^2} P(Y = y)}{\sum_{l=1}^{K} \frac{1}{\sqrt{2\pi}\sigma_k} e^{-\frac{1}{2\sigma_k}(x-\mu_l)^2} P(Y = l)} \tag{3}$$

Taking the logarithm of Equation 3 gives us

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(P(Y = k)) \tag{4}$$

Here $\delta_k(x)$ is called the discriminant function and μ_k is still the class specific mean vector, however, we assume that each μ_k shares a common variance σ^2. We then need estimates for μ_k, σ^2, and $P(Y = k)$ for this we use

$$\begin{aligned} \hat{\mu_k} &= \frac{1}{n_k} \sum_{i:y_i=k} x_i \\ \hat{\sigma^2} &= \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu_k})^2 \\ P(K = k) &= \frac{n_k}{n}. \end{aligned} \tag{5}$$

To implement LDA for a given data set each $\hat{\mu_k}$ is computed and stored. Then $\hat{\sigma^2}$ is computed using all the $\hat{\mu_k}$ and is stored. To classify a x K different discriminant functions $\delta_k(x)$ corresponding to the K different classes are computed and then x is classified as the k that produces the largest $\delta_k(x)$. This is quite limited since most classification problems have more than one attribute or predictor that is useful for classifying the response variable. We also need a way to implement LDA on problems with more than one attribute. Fortunately we can take Equation 1 and instead of using the PDF of an univariate normal distribution to find $P(X = x|K = k)$ we can can use a multivariate normal distribution to find this probability. The given X is a random variable that follows a normal distribution $X \sim N(\mu, \mathbf{\Sigma})$ the PDF of the multivariate normal distribution is given by:

$$P(X = x|K = k) = \frac{1}{(2\pi)^{p/2}|\mathbf{\Sigma}|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T\mathbf{\Sigma}^{-1}(x-\mu)} \tag{6}$$

where x is a p dimensional vector, μ is a mean vector of X specifically $E(X) = \mu$, $\mathbf{\Sigma}$ is a $(p \times p)$ covariance matrix for X. We can plug Equation into Equation to get a discriminant function with which you can calculate the probabilities of each class.

$$\delta_k(x) = x^T\mathbf{\Sigma}^{-1}\mu_k - \frac{1}{2}\mu_k^T\mathbf{\Sigma}^{-1}\mu_k + \log P(Y = k) \tag{7}$$

To compute μ_k we do the same averaging process (as in Equation 5) for each class but this time it is done element wise in a vector, $P(Y = k)$ is computed in the same way, and the covariance matrix $\mathbf{\Sigma}$ is computed by

$$\hat{\mathbf{\Sigma}} = \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu_k})(x_i - \hat{\mu_k})^T/(N - K) \tag{8}$$

To implement LDA in python I created a LDA class which has many class variables that store all data and statistics needed to compute the discriminant function and classify a point. The information stored in the class variables are:

1. Matrix storing each of the x_i in the training data
2. Array storing the response y_i in the training data
3. Number of samples

4. Number of predictors
5. An array of the counts of each of the classes
6. An array with the estimated means for each of the classes
7. The estimated variance or covariance associated with the data

The initialization method takes inputs matrix X and array y and initialized all the class variables appropriately and calls methods to estimate the means and populate the variable storing the means information, estimated the variance or covariance as well as the class probabilities. There is also a discriminant class function that returns the value of the discriminant function given a data point x_i and class k as inputs. This is called in the classify function where for an input x_i, k discriminant functions and values are computed, and the function returns the class for which the corresponding discriminant function returned the highest value. After initialization of the LDA class as an object the classify function is meant to be called manually by that function. With LDA being a relatively simple method the implementation went fairly smooth with the one hiccup being finding a way to compute $\hat{\mathbf{\Sigma}}$.

3.4 Cross Validation

To train a model there needs to be data to train and fit the model as well as test-data to provide a baseline for how the model will perform on data the the model hasn't seen during training. More specifically the data that is used to train the model is call the training data set and the data that is used to check to performance on unseen data is called the validation data set or the test data set. The validation data set is used to assess whether or not the machine learning model is being over fitted, that is that the model performs significantly better at regressing or classifying the training data than it is at doing the same for the test data. In most real situations data is hard to come by, finding enough quality data is hard and models perform much better when they see more data.

To construct a validation set all the data is taken and randomly split so no class or section of regression responses shows up disproportionately in either the training or validation set. Typically this is done in a 20:80 split where 20 percent of the data goes to the validation set and 80 percent of the data goes to the training set.

The flaw in this approach is that there is randomness; every time this is done there is a new training set and a new validation set that can look much different from the last and hence yield different errors and models. Cross validation aims to mitigate this by splitting the data set up into k distinct folds. A single fold is left out and used as the validation set and the model is trained with the remaining $k-1$ folds. This is done k times, each time leaving out a different fold. This will in turn produce k different validation errors which we can then use as one error by averaging them giving us an validation error of

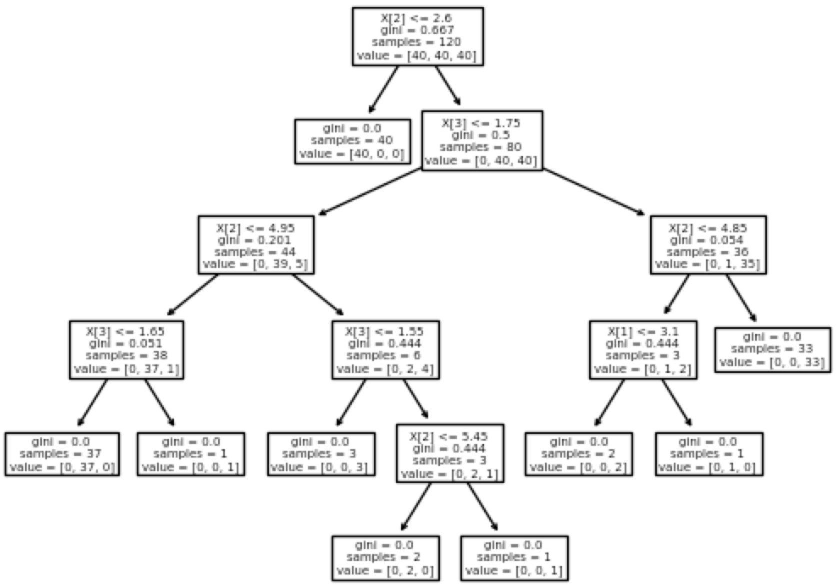
$$\text{err} = \frac{1}{n}\sum_{i=1}^{k} \text{err}_i \tag{9}$$

Here err_i denote some measure of error for the i^{th} fold of data. Typically 5 or 10 folds are used. To implement this the order of all the data is randomized and a function returns both subsets corresponding to the i^{th} fold and a subset corresponding to that fold's complement. This was easy to implement as it was only simple array manipulations thus provided no problems.

3.5 Decision Trees

Decision trees are quite versatile and can be used for either classification or regression problems. Given the structure of a decision tree, decision trees have one of the highest interpretablity of any other classifier meaning that is easy to see a direct connection between the decisions it made and what it classified the input as. A decision tree consists of nodes, a root, branches, and terminal nodes or leafs. The root is a special node at which the tree starts, meaning it has nothing leading to it or in other words it has no parent. Almost all decision trees are binary, meaning that given any non-terminal node, they can only have two children; that is two branches leading to two separate nodes. Each non-terminal node has decision information associated with it; both an attribute of a data point to make a decision on and a value to make the decision. More specifically, if the attribute is greater than or equal to the value go to the right child and if it is less go the left child. To get a prediction for an observation it starts at the root and follows the tree to a terminal node based on the attributes and values in each node. At the terminal node a prediction will be made whether it is a classification of some class or an estimation y for some regression problem.

Figure 2: Binary Decision Tree on IRIS data set



X[2] <= 2.6 gini = 0.667 samples = 120 value = [40, 40, 40]
gini = 0.0 samples = 40 value = [40, 0, 0]
X[3] <= 1.75 gini = 0.5 samples = 80 value = [0, 40, 40]
X[2] <= 4.95 gini = 0.201 samples = 44 value = [0, 39, 5]
X[2] <= 4.85 gini = 0.054 samples = 36 value = [0, 1, 35]
X[3] <= 1.65 gini = 0.051 samples = 38 value = [0, 37, 1]
X[3] <= 1.55 gini = 0.444 samples = 6 value = [0, 2, 4]
X[1] <= 3.1 gini = 0.444 samples = 3 value = [0, 1, 2]
gini = 0.0 samples = 33 value = [0, 0, 33]
gini = 0.0 samples = 37 value = [0, 37, 0]
gini = 0.0 samples = 1 value = [0, 0, 1]
gini = 0.0 samples = 3 value = [0, 0, 3]
X[2] <= 5.45 gini = 0.444 samples = 3 value = [0, 2, 1]
gini = 0.0 samples = 2 value = [0, 0, 2]
gini = 0.0 samples = 1 value = [0, 1, 0]
gini = 0.0 samples = 2 value = [0, 2, 0]
gini = 0.0 samples = 1 value = [0, 0, 1]

The tree that we will grow will be a binary tree with attributes and values at each node that decide what branch the data point being predicted will go down. Finally when a data point reaches a terminal node the predicted value of the data point is that of the value associated with that node. For a classification problem the prediction value associated with a terminal node is the most occurring class among the training instances that fall into this node of the tree. For a regression problem the prediction value associated with a terminal node is an average of all the response variables of all the training instances that fall into this terminal node. This value is set when creating the tree. To create one of the trees from a set of training instances, there is the heuristic of first growing out a large tree that performs very well on the training data. We can grow a tree by continuing to add on further decisions and pushing the training error arbitrarily close to zero. However this means that we are compromising performance on unseen data the model hasn't yet seen. So there is the concept of pruning the tree back to a smaller subset of the original un-pruned tree where it yields the best performance on real data.

3.5.1 Regression

Regression trees are a special type of Decision trees that model regression problems, that is, they aim to estimate some continuous output based on the attributes of a given input. To make a regression tree we want to stratify the predictor space into multiple non-overlapping regions in which all observations that fall into that certain region are given the same prediction, namely they are assigned the average value of all the response variables Y_i in the training set that fall into this region. Our predictor space is the set of all possible values that an observation $X = (X_1, X_2, \ldots, X_p)$ can take, and the J non-overlapping regions are denoted by $R_1, R_2, \ldots, R_J$.

To do this we take a top-down greedy approach to building the tree. This is top down in the sense that we build the tree from the root moving towards the leaves. Its also greedy in the sense that when choosing an attribute X_i to split on and a value s to decide that split, the X_i and s that are chosen is the combination that produces the lowest combined prediction error in the two resulting nodes. After the split there will be two nodes with subsets of the data in the parent node that looks like $R_{left}(j,s) = \{X|X_j < s\}$ and $R_{right}(j,s) = \{X|X_j \geq s\}$. We will need a measure of the cumulative error between each of the training y_i that have fallen into the region and prediction of the instances that fall into this region $\hat{y}_j$. The measure of error we use is the residual sum of squares (RSS) given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_j)^2 \tag{10}$$

So the process of growing the tree looks like:

1. Split the training set based on every different attribute j and every different corresponding value of x_j from the training data.
2. Record the RSS values for each of the of two regions from the split in **1.**
3. Search through all the pairs of RSS values and choose and record the attribute j and split value s that resulted in the lowest of RSS values and assign these values to the node and create a right and left child.
4. Repeat steps **1.**, **2.**, and **3.** on each of the two child nodes.

The steps **1.** through **4.** are repeated recursively on each new node until some stopping condition is met. The stopping conditions can be one or a combination of either maximum depth of a tree or minimum number of training observations in any given region R_j. These stopping conditions are chosen generously to create a large tree.

Since the tree has been grown this large there are likely many decision nodes near the bottom or leafs of the tree that specifically benefit small improvements in reducing the error for the training data however these decisions may not reflect actual trends in unseen data and may need to be removed to reduce the test error. In more general terms the tree must be pruned back to the point where it performs optimally on the data that the model hasn't seen during cross validation.

To prune the tree, the process of obtaining a sequence of optimal subtrees must be developed first. The idea behind obtaining this sequence is cost-complexity pruning where nodes are successively pruned according to which one is the weakest link. A link occurs right after an internal node but before the decision is made to split in to new nodes. When pruning a link all data that falls into that node will stop there and be classified and all further decisions in the tree are cut off and disregarded. We would like a list of z successive subtrees such that

$$T_{\alpha_1} \preceq T_{\alpha_i} \preceq \cdots \preceq T_{\alpha_z} \tag{11}$$

where $\preceq$ means that "is a subtree of" and where α is a tuning parameter that controls how much a model is penalized for having a higher complexity or a higher number of nodes. First the measure of cost error in any subtree is given by

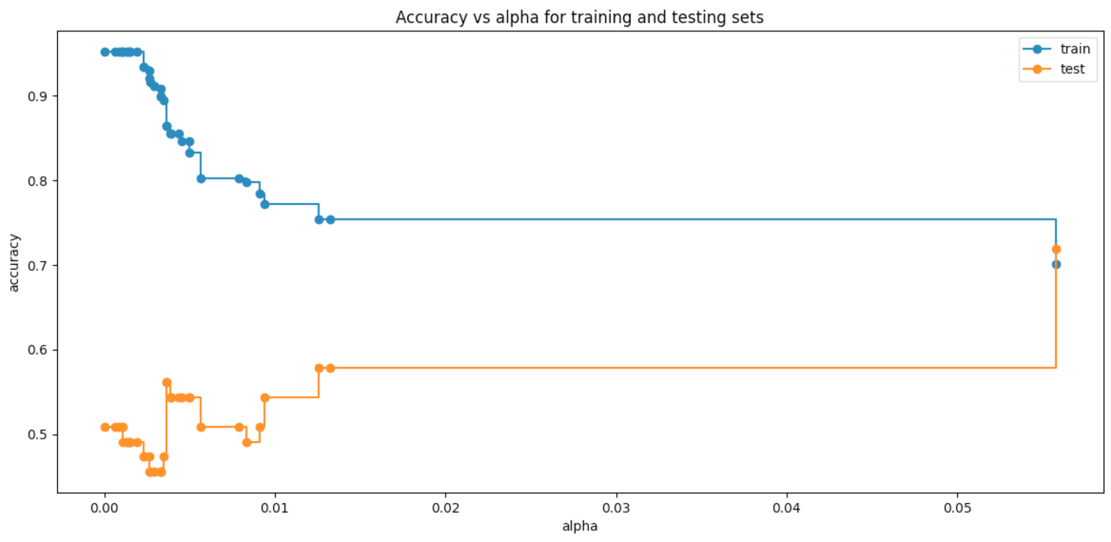
$$R(T) = \sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 \tag{12}$$

Here $|T|$ denotes the number of terminal nodes or leaf in T. This metric doesn't factor in error for high complexity, as you create more regions R_m the error will approach 0 and eventually reach that if there is only one observation in each region. So we define a new metric that takes complexity into account called the cost-complexity error

$$R_\alpha(T) = R(T) + \alpha|T| \tag{13}$$

Here the tuning parameter α controls how much the number of terminal nodes affects the error. So for each α there is a subtree $T(\alpha)$ of the full tree that minimizes $R_\alpha(T)$. So when $\alpha = 0$ there is no penalty and the tree that minimizes $R_\alpha(T)$ is the full one. As α increases the price to pay for a more complex tree also increases and smaller and smaller successive trees minimize $R_\alpha(T)$ until eventually for large enough α, $R_\alpha(T)$ is minimized by the subtree that is just the root (i.e no decisions are made and there is one classification or regression prediction). First we need to find T_{α_1} the tree corresponding to $\alpha_1 = 0$. T_{α_1} is the subtree of the $T(max)$ (the full tree) such that $R(T_0) = R(T(max))$ holds, to do this we find all nodes t that have terminal nodes as children say t_{right} and t_{left} where $R(t) = R(t_{right}) + R(t_{left})$, in this case we can just remove those two terminal nodes since they don't contribute anything.

Figure 3: Binary decision tree tuned by alpha on the cancer data set



Now the goal is to both prune a node to obtain a new tree and find the corresponding value of α for that tree. Then repeat this process until just the root is left. For any non-terminal node t we define

$$R_\alpha(t) = R(t) + \alpha \tag{14}$$

and for any corresponding branch T_t of that node t we define

$$R_\alpha(T_t) = R(T_t) + \alpha|T_t| \tag{15}$$

As α increases so does $R_\alpha(T_t)$ and $R_\alpha(t)$, however $R_\alpha(T_t)$ increases faster than $R_\alpha(t)$ meaning that for some α we will have that $R_\alpha(T_t) = R_\alpha(t)$. To find out the point at which the two values are equal we can solve the equation for α to get

$$\alpha = \frac{R(t) - R(T_t)}{|T_t| - 1} \tag{16}$$

We can now define $g_i(t)$ for $t \in T_i$ where T_i is the i^{th} subtree of T_{max}

$$g_i(t) = \begin{cases} \frac{R(t)-R(T_t)}{|T_t|-1} & ; \quad \text{t is a non-terminal node} \\ \infty & ; \quad \text{t is a terminal node} \end{cases} \tag{17}$$

The weakest link that we want to prune is the $t^* \in T_{\alpha_i}$ that achieves minimum of $g_i(t^*)$. We start with our tree T_{α_1} where $\alpha_1 = 0$ and find the node t^* that minimizes $g_1()$. Then prune the branches from that t^* to yield to subtree T_{α_2} where $\alpha_2 = g_1(T_{\alpha_1})$. Repeat this process until just the root node is left recording the values of the subtrees T_{α_i} and tuning parameter α_i throughout the process.

To actually prune the tree first the large tree is grown using the method described above using all of the data in the training set. We set this tree aside for the moment and use cross validation to grow k separate trees with the j^{th} tree being trained on every fold but the j^{th} fold. For each of these k trees a sequence of subtrees is obtained: $T_{\alpha_0}, T_{\alpha_1}, \ldots T_{\alpha_z}$ where T_{α_0} is the full tree and T_{α_z} is a tree just containing the root. For every tree in each of the k sequences the RSS is computed and denoted RSS_{j,α_i}. Then all the the RSS values are averaged for each value of α_i so that there are z RSS errors corresponding to z different values of α_i. The α_i value corresponding to the lowest RSS is chosen and used as the tuning parameter to prune the tree we set aside at the beginning.

3.5.2 Classification

Using a decision tree to do classification instead of regression is almost identical to the process of regression, the one difference being in the functions used to record the error. In regression RSS is a measure of the error, it takes the squared sum of the differences between each y_i and $\hat{y_i}$. However in classification the response variable y_i is a number that represents a certain class and otherwise has no numeric interpretation. Since the difference between two classes doesn't make sense as a metric, the obvious metric is the proportion of y_i in a region that are not of the class $\hat{y}_{R_m} = k$ for that region. Let

$$\hat{p}_{mk} = \frac{|\{y_i \in R_m : y_i = k\}|}{|R_m|} \tag{18}$$

where k denotes a specific class and the total number of classes is K. Given $\hat{p}_{mk}$ the error is given by

$$E = 1 - \max_k(\hat{p}_{mk}) \tag{19}$$

However in practice this error rate produces errors that are too small and are not sensitive enough so instead its preferred alternatives are the gini index given by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{20}$$

and the entropy given by

$$G = -\sum_{k=1}^{K} \hat{p}_{mk} \text{log} \hat{p}_{mk} \tag{21}$$

Implementing regression and classification trees was quite a larger and more complicated task than either the implementation of LDA or cross validation. To do this I implemented a node class that could store the following information:

1. Whether or not the node was a right or left node
2. The parent node
3. The right node
4. The left node
5. The decision value
6. The attribute that decision is made on
7. The RSS or Gini index error of this node
8. The RSS or Gini index error for the subtree coming from this node
9. The number of terminal nodes in the subtree under this node.
10. The prediction value if the node is terminal

There is then a decision tree class that contains general class variables to store things such as the data to train on, stopping criterion including maximum depth or minimum region size, the root which is a node object, and more. This class contains all the functions needed that are the same for both the regression and classification versions of decision trees. The functions that are specific to both regression and classification trees are declared and are left to be be overridden in a subclassed regression or classification class. This means that there are two subclasses, regression and classification, that inherit all the information from the superclass (the decision tree class mentioned earlier) but have the chance to add new functions and override (change) the superclass function to work more specifically for its task. This was essential because of the difference in the respective error functions.

The first major problem encountered when implementing this came about in the recursion. The method to grow the tree was recursive meaning that the function calls itself with new parameters: a the child node, the training information that falls into this node, and the depth of the tree. This happened specifically to make a split at the root and add a right node and a left node, then the same thing needed to be done for each of the new nodes, so this function was called again twice each with each separate node and different parameters. This recursion continues until some stopping condition is met where the function won't call itself again. The problem is this condition can be difficult to define and in my case it wasn't being met and function was infinitely recursing and crashing my computer. It took me going back and spending a lot more effort more rigorously defining the exit conditions for the function the actually work and create a tree.

Secondly pruning was difficult to figure out not only because of the programming challenges that came with it but also understanding the mathematical process of weakest link pruning and finding a resource that explained it adequately. Specifically implementing it was difficult because unlike a language like C++ where pointers exist and allow one to easily keep track of and delete nodes in python they don't exist so instead either the node has to be searched for or directions to the node must be recorded in the first place. This proved to be an extremely challenging.

3.6 Bagging

Bagging tries to address the draw back of high variance in decision trees, that is two decision trees trained on two different samples from different populations might look vastly different because of the nature of decision trees. One way to fix this problem is through Bagging. The idea of bagging is to take B different bootstrap samples from the training set and use them to create B different decision trees $T_b(x)$. In the case of regression these B different trees would be used to approximate the response variable and the predictions would be averaged together.

$$T_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x) \tag{22}$$

In the case of classification the B different classification trees would be created using bootstrap samples and instead of averaging, a voting would take place where the final prediction would be the mode of all the individual predictions.

3.7 Random Forests

Random forests are very similar to the idea of bagging except that each of the B models are only allowed m of the attributes to make splits on. That is when growing the tree instead of considering all combinations of possible attributes to split on and the respective possible values of those attributes, only a combination of m random attributes and their respective possible values can be split on. This way seems counter-intuitive however many less accurate models make for a more reliable model that has even less variance than the bagging method when it comes to making consistent predictions for different models trained on similar data. Typically in practice $m = \sqrt{p}$ where p is the number of attributes of a input variable x.

3.8 Minimal Margin Classifier

The idea behind a minimal margin classifier is that it assumes that two classes can be separated by a hyperplane and once this hyperplane is found then it can be used to classify either class 1 or class 2 depending on which side of the hyperplane a point lies. A hyperplane is defined by the equation

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0 \tag{23}$$

Where we the β_i are tuning parameters to change the boundary of the hyperplane. We can then let this hyperplane be our classifier:

$$f(x) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p \tag{24}$$

If $f(x)$ is positive then $\hat{y}$ = class 1 where as if $f(x)$ is negative then $\hat{y}$ = class 2. The problem that remains is to construct this hyperplane, to do this we place the line so that it maximizes the margin between itself and the nearest data points. These data points that the margin touches are are called support vectors.

Consider n training observations $x_1, x_2, \ldots, x_n$ with $x_i \in \mathbb{R}^p$ with corresponding $y_1, y_2, \ldots, y_n$ where $y_i \in \{-1, 1\}$ depending on whether is class 1 or class 2. Given M is the distance between the support vectors and the decision boundary then the problem becomes to find $\beta_0, \ldots, \beta_p$ that maximize the margin M under the constraints

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{25}$$

$$y_i(\beta_0 + \beta_1 X_i 1 + \beta_2 X_i 2 + \cdots + \beta_p X_i p) \geq M \quad \text{; for all } i = 1, .., n \tag{26}$$

However this only works in the case that this hyperplane exists. In most real situations data isn't separated this nice and distinctly.

3.9 Support Vector Classifier

The Support Vector Classifier (SVC) addresses the problem of not being able to work in the case where a hyperplane cannot be found, more specifically even for the most ideal hyperplane some points will lie on the wrong side and be incorrectly classified. To do this SVCs have soft margins where we can allow a certain amount data points to be on the wrong side of the hyperplane. To do this we introduce the concept of slack variables ϵ_i that provide a buffer for points to be on the wrong side of the margin. Then the problem becomes for $\beta_0, \ldots, \beta_p$ and $\epsilon_1, \ldots, \epsilon_p$ to maximize M under the constraints

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{27}$$

$$y_i(\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip}) \geq M(1 - \epsilon_i) \tag{28}$$

$$\sum_{i=1}^{n} \epsilon_i \leq C \quad \text{where } \epsilon_i \geq 0 \tag{29}$$

We can see that if $\epsilon_i > 1$ then the the point lies on the wrong side of hyperplane so conceptually C is the maximum number of points that can lie on the wrong side of the and is then used as a tuning parameter to control this. Also the hyperplane only depends on the data points that will lie in the margin or on the wrong side of the margin, in other words the hyperplane only depends on the support vectors. This means that when solving the optimization problem it is much more computationally efficient since the number of support vectors will be much smaller than n. The solution to the optimization problem involves quadratic programming and language primal functions which is out of the scope of this project.

3.10 Support Vector Machine

SVCs can only produce linear boundaries because of the constraint of the hyperplane only involving first degree terms. Any combination of higher degree terms could lead to more complex and versatile decision boundaries. The solution to the SVC boils down to

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle \tag{30}$$

where S is the set of support and $\langle x, y \rangle$ denote the inner product between x and y defined by $\sum_{i=1}^{p} x_i y_i$. Simplifying this equation we see that this does in fact yield the hyperplane defined in Equation 24.

$$\begin{aligned}
f(x) &= \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle \\
&= \beta_0 + \alpha_1 \langle x, x_1 \rangle + \alpha_2 \langle x, x_2 \rangle + \cdots + \alpha_s \langle x, x_s \rangle \\
&= \beta_0 + \alpha_1(x_1 x_{11} + x_2 x_{12} + \cdots + x_p x_{1p}) + \cdots + \alpha_s(x_1 x_{s1} + x_2 x_{s2} + \cdots + x_p x_{sp}) \\
&= \beta_0 + x_1(\alpha_1 x_{11} + \alpha_2 x_{21} + \cdots + \alpha_s x_{s1}) + \cdots + x_p(\alpha_1 x_{1p} + \alpha_2 x_{2p} + \cdots + \alpha_s x_{sp}) \\
&= \beta_0 + x_1 \beta_1 + x_2 \beta_2 + \cdots + x_p \beta_p
\end{aligned}$$

However this is still linear if we replace the inner product with a function K that determines the similarity between x and x_i which is know as the kernel allowing us to construct non-linear decision boundaries.

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x, x_i) \tag{31}$$

Now we can make any polynomial decision boundary of degree d by defining the kernel as

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij}x_{i'j})^d \tag{32}$$

There are many other different kernels that be defined and used to adapt the support vector machines.

Due to my lack of knowledge of optimization and the fact that hand coding a program that maximizes convex quadratic programming is quite complex and outside of the scope of this project I didn't write a python implementation of support vector machines, but rather used an implementation through a python library called "Sci-kit Learn".

4 Results

4.1 Wine Quality

To assess the performance of models on the wine quality data set only two of all the algorithms were used because the rest are suited for classification instead of regression. The absolute error here is defined to be the average distance between the prediction of the wine quality and the actual rating, so on average the regression tree made predictions that were on average 0.51729186 off from the actual rating. Here the random forest had 100 estimators in each with a maximum depth of 4, and the regular tree had its maximum depth set to 5.

Algorithm	Error
Regression Tree	0.51729186
Random Forest	0.50931437

Considering that the wine was rated on a 10 point scale / 0.5 points off the prediction is a respectable error for a model to make. The ratings in the data set were also discrete meaning they only took on integer values.

4.2 IRIS

The first time training on the IRIS data set was done with 80 percent of the data used for training and 20 percent for testing. With this train test split all models were achieving 100 percent test accuracy. This didn't provide useful information as to what model was performing the best so I considered a harder problem by switching the test and training data; training the data on 20 percent of the data set and testing on the remaining 80 percent. This is a much harder problem because the models have a lot less data to try and learn the overall patterns in the population. This is reflected in the fact that the error percentages weren't all zero. Here the random forest had 100 different estimators each with a maximum depth of 2. The bagging classifier had 10 trees each with a no maximum depth. Finally the regular tree had no maximum depth.

Algorithm	Error
LDA	5.0
SVM	8.333
Classification Tree	5.833
Tree W/ Bagging	5.0
Random Forest	6.667

The errors above were calculated using the form $\frac{\text{(Total number)}-\text{(Number correct)}}{\text{(Total number)}} * 100\%$
Here it isn't clear that there were any standout performers from the others because every model performed decently well although the Linear Discriminant Analysis and Classification Tree with Bagging models performed the best. This is evidence not to always jump to using the most complicated model first because it might not be the most appropriate or best model, and speed and interpretablity is always preferable if the model offers the same performance.

4.3 Banknote Authentication

The models trained on the Banknote Authentication data sets were trained on 80 percent of the full data set and tested on the remaining 20. Here the random forest had 100 different estimators each with a maximum depth of 5. The bagging classifier had 100 trees each with a no maximum depth. Finally the regular tree had no maximum depth.

Algorithm	Error
LDA	0.0
SVM	0.0
Classification Tree	1.832
Tree W/ Bagging	0.366
Random Forest	0.366

The results here are interesting because all of the tree based models had higher error rates than the two models that used some kind of decision boundary. This implies that that there is likely some boundary in that 4 dimensional attribute space that provides a near perfect or perfect boundary to classify on. Tree based methods produce boundaries in this space but they are rectangle-like in the sense the border can only be drawn by is $s > j$ or is $s \leq j$ and this might be the short coming of the tree based methods in this situation.

4.4 Breast Cancer

Again in the Breast Cancer data set all models were trained on 80 percent of the full data set and tested on the remaining 20. Here the bagging classifier had 100 trees each with a maximum depth of 1 and the regular tree had no maximum depth.

Algorithm	Error
LDA	35.0877
SVM	35.0877
Classification Tree	43.860
Tree W/ Bagging	42.105
Random Forest	19.298

Here its easy to see this is a harder problem where there likely isn't a perfect solution to this because there are many variables that may play a small role in whether or not there is a recurrence of events but are not included. There is a clear winner for best performing model, specifically the random forest. This is interesting in particular because the parameters that were used to achieve the lowest possible error with this model seem surprising. There were 100 separate tree estimators each with a maximum depth of one, meaning the 100 separate estimators each making a decision based on one attribute and voting on whether or not to classify a recurrence of events. This is likely because there was one or a couple of attributes that were much more important than the others for making this decision and this model never took into account the less important attributes.

5 Conclusion

Through this project I dove into the mechanics of several machine learning algorithms and techniques including linear discriminant analysis, cross validation, both regression and classification trees, bagging, random forests, and support vector machines. Through this I also spent much of my time implementing these algorithms in python which was further increased my understanding of the algorithms and my retention of their workings. I then used these implementations to explore their performance on various data sets and look into how different parameters affect the performance of the model on the data. This is valuable knowledge and experience going forward and I will be able to use these algorithms to make predictions in the future when I am presented with these types of problems.

The biggest limitations in this project were time and knowledge in other mathematical disciplines. Time is a factor that hardly ever plays to your advantage especially in a 10 week time frame where study is independent. If given more time I could have covered more algorithms delving more in to regression algorithms instead of mainly classification ones. The other thing that was a limitation was my lack of knowledge in other mathematical subjects, specifically optimization as this made it quite difficult to understand on a higher level how support vector machines work.

I think that there are some implicit assumptions of normality in attributes when used as random variables that don't necessarily lead to the most theoretically accurate model but its hard to break way from the assumptions because failing to accept the assumptions in turn breaks the model. Also each of the attributes in a input variable seemed to be of equal importance in almost all methods with the exception of decision trees. This seems like something that might be useful to break way from since it could place more emphasis on variables that matter more.

In the future after exploring more and different machine learning algorithms in depth I would like to spend more time looking at other more specialized algorithms to specific data sets. Eventually I would like to build and experiment off of other already existing algorithms without pressure to succeed or fear of failure; through this I think I could learn and explore new things that I wouldn't normally think to ask during a scheduled project with hard deadlines. This would most likely result in me learning what didn't work rather than what did, but I think that I have seen through this project that that as important.

All code from this project can be found at https://github.com/qbock

References

[1] Fisher R. (2019). *UCI Machine Learning Repository: Iris Data Set.* [https://archive.ics.uci.edu/ml/datasets/Iris]. Irvine, CA: University of California, School of Information and Computer Science.

[2] Cortez, P., Cerdeira, A., Almeida, F., Matos, T., and Reis, J. (2019). *UCI Machine Learning Repository: Wine Quality Data Set.* [https://archive.ics.uci.edu/ml/datasets/Wine+Quality]. Irvine, CA: University of California, School of Information and Computer Science.

[3] Lohweg, V. (2019). *UCI Machine Learning Repository: banknote authentication Data Set* . [https://archive.ics.uci.edu/ml/datasets/banknote+authentication]. Irvine, CA: University of California, School of Information and Computer Science.

[4] Zwitter, M. and Soklic, M. (2019). *UCI Machine Learning Repository: Breast Cancer Data Set* . [https://archive.ics.uci.edu/ml/datasets/Breast+Cancer]. Irvine, CA: University of California, School of Information and Computer Science.

[5] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2017). An introduction to statistical learning: With applications in R. New York, NJ: Springer.

[6] Hastie, T., Friedman, J., and Tisbshirani, R. (2017). The Elements of statistical learning: Data mining, inference, and prediction. New York, NJ: Springer.