CSE 601: Data Mining and Bioinformatics

Project 3: Classification Algorithms

By,

Shri Sai Sadhana Natarajan (50247664)

Sneha Parshwanath (50248890)

Soumya Venkatesan (50246599)

K Nearest Neighbours

Introduction:

K Nearest Neighbour (KNN) algorithm is a non-parametric method used for data classification that estimates how likely a data point is to be a member of a group based on what groups the data points nearest to it are in. It is called non-parametric because it doesn't make any assumptions on the underlying data distribution. An object is classified in KNN based on majority vote, that is the most common class label among all of its k nearest neighbours is assigned to it. KNN is a lazy learner algorithm and does not generate a dataset model beforehand and all the computation will be deferred until classification.

Implementation:

1. The data file and the number of nearest neighbours to be considered (k value) is obtained as input.
2. The data is read from the given input file and is checked for the presence of nominal attributes (categorical data given as strings) in the columns. If any nominal data is found, it is replaced with unique integer values where each integer represents a nominal value. Finally, a data array containing both nominal and continuous values in float data type is formed.
3. 10-fold cross validation is performed on the dataset to split it into train and test sets. In every round of cross validation, 9 parts of the dataset is used a train set and 1 part is used as test set.
4. After splitting the data, mean and standard deviation is calculated for the training set to help in normalization. Then, the train and test sets are normalized by applying z-score normalization. z-score normalisation is calculated as follows:

   $$\text{Normalised}(x_i) = (x_i - \text{mean}) / \text{standard deviation}$$

5. For every testing record, the Euclidean distance to every training record is computed and stored in a list.
6. The list of distances is sorted and the top K nearest neighbours are chosen and stored.
7. Then, the class label for the testing record that is currently unknown is chosen based on the majority class label that the K nearest neighbours correspond to. This label is stored as the predicted label.
8. After predicting the label for every testing record in the test set, they are matched with the original test set labels and performance metrics: accuracy, precision, recall and F-1 measure are computed. They are calculated as follows:
   - Accuracy: (true positive + true negative) / (true positive + true negative + false positive + false negative)
   - Precision: true positive / (true positive + false positive)

- Recall: true positive / (true positive + false negative)
- F-1 measure: (2 * recall * precision) / (recall + precision)

9. Now, a different test set from the remaining parts of the dataset that was not previously used as test set is picked and steps 4-8 are repeated. This process is continued until each of the 10 parts of the dataset is used as test set once.
10. The final measure is computed by taking the average of all performance measure values calculated at each of the 10 folds.

Choice Description:

K value:

k value determines the number of neighbours that should be considered while assigning a class label to a test record. If k is too small or too large, the result will be sensitive to noise point or data might be misclassified. So it is very important to choose an optimal value for k.

Choosing an optimum k value is done by inspecting the data. Cross validation is also a good way to determine a good k value by considering different data results and using independent dataset to validate the K value. Several past researches have shown that an optimum value of k usually lies between 3-10 for most datasets. Also k is generally chosen to be an odd number to avoid ties. Sometimes, it is said that square root of the number of records in the train set is a good value for k. But the best way is to do cross validation and try various k values to determine the optimum k value.

Using cross validation and trying various k values, k = 5 was observed to be an optimal value.

Distance calculation:

The k nearest neighbours is determined by calculating the distance between two data points. The best method to calculate distance between the two data points is using Euclidean distance measure. Euclidean distance works very well for a dataset containing only continuous attributes. But if the dataset contains categorical attributes, hamming distance measure is much preferred since it computes binary vectors.

However, in this project categorical attributes are also converted into numerical values and Euclidean distance measure is used for calculating distance. euclidean() function from scipy.spatial.distance library is used for Euclidean distance calculation. Formula to calculate Euclidean distance is:

$$d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$$

Continuous and categorical attributes:

Categorical attributes if present in the dataset, are replaced with unique integer values where each integer represents a categorical value. The mean and standard deviation are computed and all the values are normalised using z-score normalization. The normalised categorical and continuous values will be used in the computation.

Result:

For Dataset: project3_dataset1.txt

k - value	Accuracy	Precision	Recall	F1-measure
3	0.964285714	0.977294197	0.924327339	0.949300097
5	0.966071429	0.977719503	0.928872794	0.953625933
7	0.958928571	0.977565657	0.909632775	0.940638616
9	0.966071429	0.986052836	0.922928738	0.952114844
10	0.971428571	0.990598291	0.921177896	0.953191429
15	0.966071429	0.990277778	0.920623636	0.953566664
20	0.955357143	0.989769821	0.892607208	0.937974175

For Dataset: project3_dataset2.txt

k - value	Accuracy	Precision	Recall	F1-measure
3	0.665217391	0.537596848	0.447636399	0.468691372
5	0.669565217	0.538409091	0.394754193	0.429722727
7	0.656521739	0.511722488	0.378423944	0.42089418
9	0.689130435	0.596099734	0.385251751	0.448775729
10	0.701508712	0.626314192	0.406031666	0.470795664
15	0.710869565	0.641181319	0.392305845	0.47180119
20	0.704347826	0.653751804	0.369595463	0.447977328

Evaluation:

- KNN is a distance based algorithm that classifies data based on the distance between them and the distance between two continuous attributes can be clearly defined using Euclidean distance. Dataset 1 contains only continuous attributes and hence KNN gives the highest accuracy for dataset 1 compared to other implemented classification algorithms.
- Dataset 2 contains continuous and nominal attributes. The distance between two nominal attributes is always one and hence KNN does not perform well on datasets containing nominal attributes. Thus, the accuracy of KNN for dataset 2 is lesser compared to other implemented classification algorithms.
- Another reason for KNN being less accurate on dataset 2 is that it is smaller in size and hence there are not enough examples to train the classifier to classify accurately
- It can also be observed that the value of k has a huge impact on the accuracy of the dataset. If k is too small, the classifier could be sensitive to outliers and if k is too large, the classifier could misclassify the data.

Pros:

- KNN is easy and simple to implement.
- It can handle multiclass data.
- It doesn’t make any assumptions about the data.
- KNN responds quickly to changes in the input.

Cons:

- The value of parameter K should be determined.
- It is computationally expensive since the distance of each test sample from all training samples should be computed.
- It has a high memory requirement since it stores the entire training data.
- If the attributes are not scaled properly, the algorithm may sometimes misclassify the data.

Naïve Bayes

Introduction:

Naïve Bayes classifier is a probabilistic classifier based on Bayes theorem. This classifier tries to predict the class membership probabilities and is particularly suited when the dimensionality of inputs is high. It functions on the assumption that the attributes in a dataset are independent to each other. Bayes theorem is based on conditional probability and it describes the probability of an event based on prior knowledge of conditions that might be related to the event. It is computed using the following formula:

$$P(H_i \mid X) = \frac{P(H_i)\, P(X \mid H_i)}{P(X)}$$

Where,

$P(H_i \mid X)$ = Class Posterior Probability: Probability of hypothesis H_i holding true that X belongs to class C_i given that we know the attributes of X.

$P(X \mid H_i)$ = Descriptor Posterior Probability: Probability of evidence that there exists a X such that it satisfies the hypothesis H_i of belonging to a class.

$P(H_i)$ = Class Prior probability: Probability of hypothesis H_i holding true that is independent of the data

$P(X)$ = Descriptor Prior Probability: Probability of evidence that X exists

Implementation:

1. The data is read from the given input file, the columns containing nominal (categorical data given as string) and continuous attributes (given as real values) are identified and stored as separate sets.
2. 10-fold cross validation is performed on the dataset to split it into train and test sets. In every round of cross validation, 9 parts of the dataset is used a train set and 1 part is used as test set.
3. A count of the number of training samples belonging to each class C_i (class 0 and 1 in this case) is taken and class prior probability $P(H_i)$ for each class is calculated by dividing the number of samples belonging to a particular class with the total number of samples.
4. The descriptive posterior probability $P(X \mid H_i)$ is calculated separately for nominal and continuous attributes which is as follows:
    a. For nominal attributes, the descriptive posterior probability is calculated by counting the number of training records containing the same value as that of the testing sample for a particular class and dividing it by the number of samples in that class.

   b. For continuous attributes, descriptive posterior probability is calculated by using Gaussian distribution. For each continuous feature, the mean and standard deviation are found and probability density function is used to calculate the probability.
   c. Since each feature X is independent of the other, the steps 4a and 4b are repeated for each feature and the final descriptive posterior probability for class C_i (class 0 and 1) is obtained by multiplying the probabilities of all features for that class together.
5. The class posterior probability $P(H_i \mid X)$ for each class C_i (class 0 and 1) is calculated by multiplying the class prior probability and descriptive posterior probability calculated previously for each class.
6. The class posterior probability of both the classes is compared and the one with the highest probability value is chosen as the class label for that testing record. This label is stored as the predicted label.
7. After predicting the label for every testing record in the test set, they are matched with the original test set labels and performance metrics: accuracy, precision, recall and F-1 measure are computed. They are calculated as follows:
   - Accuracy: (true positive + true negative) / (true positive + true negative + false positive + false negative)
   - Precision: true positive / (true positive + false positive)
   - Recall: true positive / (true positive + false negative)
   - F-1 measure: (2 * recall * precision) / (recall + precision)
8. Now, a different test set from the remaining parts of the dataset that was not previously used as test set is picked and steps 3-7 are repeated. This process is continued until each of the 10 parts of the dataset is used as test set once.
9. The final measure is computed by taking the average of all performance measure values calculated at each of the 10 folds.

Choice Description:

Continuous features:

The descriptive posterior probability is usually calculated by just counting the records with the specific value. But this method cannot be used to calculate descriptive posterior probability for continuous attributes. There are two ways to overcome this:

1. Using Discretization: Continuous attributes can be discretized and replaced with values corresponding to the discrete interval, thus converting continuous attributes to categorical attributes. But if the intervals are too small or too large, misclassification of data might take place.
2. Using Gaussian distribution: The mean and standard deviation will be calculated for each continuous attribute in the given dataset, and probability density function is used to calculate the descriptive posterior probability for the given test sample.

Gaussian distribution is considered as a better method to handle continuous attributes using Naïve Bayes classifier. So Gaussian distribution is used to calculate descriptive posterior probability in this project.

Zero-probability issue:

Naïve Bayes classifier suffers from zero probability that is, if a particular attribute value that was not present in the training set occurs, it will be unable to make a prediction and the probability of that attribute will become zero. This will give zero probability to entire descriptive posterior probability regardless of the probabilities of the other features. This problem can be overcome by using Laplacian Correction, where a constant value (1) is added to each case and total number of samples is increased by the number of distinct values for the attribute.

Result:

For Dataset: project3_dataset1.txt

Accuracy: 0.9357142857142857

Precision: 0.9241315440217044

Recall: 0.9053435453435454

F-1 Measure: 0.913209052226269

For Dataset: project3_dataset2.txt

Accuracy: 0.7021739130434783

Precision: 0.5707868713673667

Recall: 0.617493091703618

F-1 Measure: 0.5842003761632245

Evaluation:

- The huge difference in the accuracy between the two datasets is because of the higher number of attributes in dataset 1 compared to dataset 2.
- It is known that Naïve Bayes classifier works on the assumption that attributes are independent of each other. But dataset 1 contains only continuous values (attributes are not independent) and hence the accuracy of Naïve Bayes for dataset 1 is lesser than the accuracy given by other classification algorithms implemented.
- But for dataset 2, Naïve Bayes classifier gives a higher accuracy compared to other algorithms. This is because dataset 2 contains nominal attributes (independent attributes) which is handled best by Naïve Bayes algorithm.

Pros:

- It is simple and easy to understand and implement.
- It is efficient when applied to large datasets.
- It is not affected by irrelevant features.
- It converges faster and hence will require only a small amount of training data to estimate the parameters.
- It can handle both continuous and discrete values.

Cons:

- It doesn’t work well with correlated data since it assumes every feature is independent. But in real life this is seldom satisfied since real-life data are mostly correlated.
- It has zero-probability issue that is, if a particular attribute value that was not present in the training set occurs, it will be unable to make a prediction and will give zero probability while calculating posterior probability.
- It is not easy to compute probabilities when the attributes are continuous in nature. Binning or Gaussian distribution should be used to compute probability and cannot be done by just frequency counting.

Decision Tree

Introduction:

Decision Trees are non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. Decision trees also provide the foundation for more advanced ensemble methods such as bagging, random forests and boosting. Decision trees are constructed via an algorithmic approach that identifies ways to split a dataset based on different conditions. Tree models where the target variable can take a discrete set of values are called classification trees. Decision trees where the target variable can take continuous values are called regression trees. Classification and Regression Tree (CART) is a general term for this.

Implementation:

1. The data is read from the given input file, the columns containing nominal (categorical data given as string) and continuous attributes (given as real values) are identified and stored as separate sets.
2. 10-fold cross validation is performed on the dataset to split it into train and test sets. In every round of cross validation, 9 parts of the dataset is used a train set and 1 part is used as test set.
3. For each of the fold, decision tree algorithm is called for the train data of that fold.
4. While constructing the decision tree, each time the best split attribute and its index are calculated for every node of the tree. To decide the best split, node impurity function is used.
5. The node impurity function used by our implementation is the weighted Gini Index. It is calculated as follows:

   $GINI(t) = (1 - \sum p(j/t)^2) * (\text{size of split/total number of data records})$

   where

   $p(j/t)$ is the relative frequency of class j at node t
6. For selecting the best split, the minimum Gini Index value is selected. A node of the tree with the minimum Gini Index and best split attribute is created.
7. If the best split attribute is categorical, the records with attribute value equal to the best split attribute value goes to the left child of that node and remaining get assigned to the right of the child node.
8. If the best split attribute is continuous, the records with attribute value less than the best split attribute value goes to the left child of that node and remaining get assigned to the right of the child node.
9. The tree grows recursively until the stop split criteria is met.

10. When the minimum number of data records becomes less than the user specified parameter, minimumRecordSize for either of the children, the node is converted to leaf node and class is assigned by majority vote of all the records at that node.
11. The node is also converted to child node and assigned class label by majority voting of all records at that node, if the user specified parameter, maximum depth is reached.
12. Once the tree is constructed, the root of the constructed tree is returned.
13. The classify test record function traverses the constructed tree recursively until a leaf node is reached and assigns the class label of the leaf node to that test record. This is repeated for all the test records in the given fold.
14. After predicting the label for every testing record in the test set, they are matched with the original test set labels and performance metrics: accuracy, precision, recall and F-1 measure are computed. They are calculated as follows:
    a. Accuracy: (true positive + true negative) / (true positive + true negative + false positive + false negative)
    b. Precision: true positive / (true positive + false positive)
    c. Recall: true positive / (true positive + false negative)
    d. F-1 measure: (2 * recall * precision) / (recall + precision)
15. Now, a different test set from the remaining parts of the dataset that was not previously used as test set is picked and steps 3-13 are repeated. This process is continued until each of the 10 parts of the dataset is used as test set once.
16. The final measure is computed by taking the average of all performance measure values calculated at each of the 10 folds.

Choice Description:

Categorical features:

Decision Tree can handle categorical features. Every split in a decision tree is based on a feature. If the feature is a categorical feature, the binary split is done with the element that belongs to the same class is added to the left of the node and the element that does not belong to the class is added to the right of the node. Splitting continues until nodes contain a minimum number of training samples or a maximum tree depth is reached.

Continuous features:

For continuous features, the splitting can be handled in different ways. It can be discretized to form an ordinal categorical attribute or binary decision. Here, binary split is done by considering all possible splits and finding the best cut. If the split node value is less than a certain threshold value it is assigned to the left child of the node, otherwise to the right child of the node

Best feature:

The split with the best cost i.e the lowest cost (because we minimize cost) is selected. All input variables and all possible split points are evaluated and chosen in a greedy manner based on the cost function which is Gini Index. The Gini cost function is used which provides an indication of how pure the nodes are, where node purity refers to how mixed the training data assigned to each node is.

Post processing:

Post pruning is trimming the nodes of the decision tree in a bottom-up fashion after growing the tree to its entirety. If the generalization error improves after trimming, then the sub-tree is replaced by a leaf node. The class label of leaf node is determined from majority class of instances in the sub-tree

Result:

For Dataset: project3_dataset1.txt

Accuracy: 0.9196428571428573

Precision: 0.8967701863354038

Recall: 0.8974912124912127

F-1 Measure: 0.8958172398719031

For Dataset: project3_dataset2.txt

Accuracy: 0.6130434782608696

Precision: 0.4233352431881844

Recall: 0.4356929985219458

F-1 Measure: 0.42064988041218954

Evaluation:

- The decision tree classification algorithm performs better with dataset 1 that has more number of attributes compared to the dataset 2.
- The algorithm followed a greedy approach and performed well with the larger dataset.
- The algorithm is able to handle both continuous and categorical values in the dataset.

Pros:

- It is simple and easy to interpret and implement.
- Performs well with large datasets.
- It handles large set of features.
- Classifying a test record is fast with a worst-case complexity of O(max_depth)
- It Can handle both continuous and discrete values.

Cons:

- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated.
- It requires pruning to avoid over-fitting data.
- It is not expressive enough for modelling continuous variables.
- A wrong decision can lead to all subsequent decisions being wrong as well.
- There are concepts that are hard to learn because decision trees do not express them easily such as XOR, parity.

Random Forest

Introduction:

Random Forest is a statistical algorithm that is used to cluster points of data in functional groups. When the data set is large and there are many variables it becomes difficult to cluster the data because not all variables can be taken into account, therefore the algorithm can also give a certain chance that a data point belongs in a certain group. In a random forest algorithm, the number of trees grown and the number of samples that are used at each split can be chosen by hand. It is also one of the most used algorithms, because of its simplicity and the fact that it can be used for both classification and regression tasks.

Implementation:

1. The data is read from the given input file, the columns containing nominal (categorical data given as string) and continuous attributes (given as real values) are identified and stored as separate sets.
2. 10-fold cross validation is performed on the dataset to split it into train and test sets. In every round of cross validation, 9 parts of the dataset is used a train set and 1 part is used as test set.
3. The number of trees to grow can be read from the user as input and the number of random features is the square root of the number of attributes of the dataset.
4. Choose the number of features (< total features) used to calculate the best split at each node.
5. For each tree, choose a training set by choosing N times with replacement from the training set and for each node choose the features randomly and calculate the best split.
6. As the last step use the majority voting among all the trees and predict the label.
7. After predicting the label for every testing record in the test set, they are matched with the original test set labels and performance metrics: accuracy, precision, recall and F-1 measure are computed. They are calculated as follows:
    a. Accuracy: (true positive + true negative) / (true positive + true negative + false positive + false negative)
    b. Precision: true positive / (true positive + false positive)
    c. Recall: true positive / (true positive + false negative)
8. F-1 measure: (2 * recall * precision) / (recall + precision)
9. Now, a different test set from the remaining parts of the dataset that was not previously used as test set is picked and steps 3-7 are repeated. This process is continued until each of the 10 parts of the dataset is used as test set once.
10. The final measure is computed by taking the average of all performance measure values calculated at each of the 10 folds.

Result:

For Dataset: project3_dataset1.txt

Accuracy: 0.9571428571428573

Precision: 0.945387676777837

Recall: 0.9427515077515078

F-1 Measure: 0.9434893034711033

For Dataset: project3_dataset2.txt

Accuracy: 0.6760869565217391

Precision: 0.5324539512774807

Recall: 0.47575075509286024

F-1 Measure: 0.4951607603762776

Evaluation:

- The random forest classification algorithm performs better for both dataset 1 and dataset 2 when compared to decision tree classification.
- The huge sample size of dataset 1 helps to achieve greater accuracy.
- Accuracy is improved on increasing the value of the number of trees but results in a slower model

Pros:

- It is simple, fast and flexible.
- Can easily handle categorical features.
- Training is efficient for large datasets and less prone to over-fitting.
- It has the ability to do multi-class classification naturally.
- It can more accurate compared to decision tree classification.

Cons:

- Random forest is not easy to visually interpret.
- It becomes slower as the number of trees to grow is increased.
- It is fast to train but can be slow to predict once trained.
- Random Forests tend to prefer splitting on variables that have a large number of unique values.

Boosting

Introduction:

Boosting is an ensemble method for improving the model predictions of any given learning algorithm. The idea of boosting is to train weak learners sequentially, each trying to correct its predecessor. It fits a sequence of weak learners on different weighted training data. It starts by predicting original data set and gives equal weight to each observation. If prediction is incorrect using the first learner, then it gives higher weight to observation which have been predicted incorrectly. Being an iterative process, it continues to add learners until a limit is reached in the number of models or accuracy. We have implemented AdaBoost algorithm.

Implementation:

1. The data is read from the given input file, the columns containing nominal (categorical data given as string) and continuous attributes (given as real values) are identified and stored as separate sets.
2. 10-fold cross validation is performed on the dataset to split it into train and test sets. In every round of cross validation, 9 parts of the dataset is used a train set and 1 part is used as test set.
3. The number of classifiers to be used for boosting is set in the parameter T.
4. To build each classifier, initially weights are assigned to all training records as 1/length of training records.
5. For each iteration of AdaBoost (until T trees are grown), boostrap sample is selected at random from the given training sample and weights data.
6. The decision tree algorithm runs for the bootstrap sample selected.
7. After classification based on the constructed tree of all the training records, the error rate/misclassification error is calculated using:

   misclassification error = sum of product of each weight times the misclassification value/ total of weights

   Where misclassification value = 1 if misclassified,0 otherwise

8. When the misclassification error is above 0.5 we reject that classifier. When it is below 0.5, we calculate the classifier importance using the formula:

   $\alpha = 0.5 * \ln$ (1- misclassification error/misclassification error)

9. The weights of each records are updated according to:

   $$w_j = w_j * \exp(-\alpha y_i C(x_i))$$

   where, w_j is the weight of jth record

   α is the value calculated in step 8

   y_i is the class label predicted for the model

   $C(x_i)$ is the actual class label

The weights are normalized and lie between values [0,1].

10. For test records, the prediction is made using the weighted average value from all the classifiers using that classifier's importance.
11. Adaboost algorithm is repeated for all k –folds.
12. After predicting the label for every testing record in the test set, they are matched with the original test set labels and performance metrics: accuracy, precision, recall and F-1 measure are computed. They are calculated as follows:
    a. Accuracy: (true positive + true negative) / (true positive + true negative + false positive + false negative)
    b. Precision: true positive / (true positive + false positive)
    c. Recall: true positive / (true positive + false negative)
    d. F-1 measure: (2 * recall * precision) / (recall + precision)
13. Now, a different test set from the remaining parts of the dataset that was not previously used as test set is picked and steps 3-7 are repeated. This process is continued until each of the 10 parts of the dataset is used as test set once.
14. The final measure is computed by taking the average of all performance measure values calculated at each of the 10 folds.

Result:

For Dataset: project3_dataset1.txt

Accuracy: 0.9303571428571429

Precision: 0.9327210745013949

Recall: 0.8742024642024642

F-1 Measure: 0.9013776561602649

For Dataset: project3_dataset2.txt

Accuracy: 0.6478260869565217

Precision: 0.4983355606381922

Recall: 0.46373722768459624

F-1 Measure: 0.4717756732411904

Evaluation:

- From the metrics we can see that boosting performs better than decision trees but not as good as random forest.
- Accuracy is improved on increasing the value of T but results in a slower model

Pros:

- It is fast and only one parameter to tune T.
- Training error drops exponentially.
- It is shown to have excellent generalization properties.
- It is flexible and can be combined with any classifier.

Cons:

- Unlike bagging and random forests, can overfit if number of trees is too large
- Susceptible to noise.
- Computationally expensive.
- Sub-optimal solutions.

References:

- ✓ Lecture Slide: Classification1, Classification2, Classification3, Classification5
- ✓ https://docs.scipy.org/doc/scipy/reference/spatial.distance.html
- ✓ https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html
- ✓ https://discuss.analyticsvidhya.com/t/how-to-choose-the-value-of-k-in-knn-algorithm/2606
- ✓ https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
- ✓ https://en.wikipedia.org/wiki/Naive_Bayes_classifier
- ✓ https://scikit-learn.org/stable/modules/tree.html
- ✓ https://towardsdatascience.com/decision-tree-in-machine-learning-e380942a4c96
- ✓ https://www.csd.uwo.ca/courses/CS4442b/L5-ML-Boosting.pdf
- ✓ https://stackoverflow.com/questions/736043/checking-if-a-string-can-be-converted-to-float-in-python
- ✓ https://towardsdatascience.com/adaboost-for-dummies-breaking-down-the-math-and-its-equations-into-simple-terms-87f439757dcf