Need to split the dataset into train and test sets.

In supervised learning, the model has to be trained with some data having labeled input with known outputs. Once the training of the model is done, the model has to be tested on the **test data which is not exposed earlier to the model during training**.

To achieve this, we need to split the dataset into train and test sets using: sklearn.model_selection.train_test_split giving required parameters.

Problem and Need: If by any case, the test data is exposed during the training of the model, it will result in **over fitting** and overestimating the accuracy of the model when the same training data is tested. This model will not generalize well on the new test data. As a result, the model will become potentially unusable to deploy for practical applications. This problem is also known as **Data Leakage**.

Solution: To analyze the generalization of the model we need to test the model with novel and unseen data. Thus, there comes a need of the test dataset.

1. In order to solve the problem of data leakage, we can divide the dataset into train-validate-test sets.
2. Thus, the validation dataset which is taken from the training set can be used as the initial test set before performing the actual testing on test dataset.
3. Secondly, the k-fold cross validation can also be used to properly split the datasets with the cross validation split strategy.

Overall, we split the dataset into train and test sets to prevent over-fitting, data leakage and isolate the test dataset so that it is not exposed to model while the model is learning.

Example:

 # splitting the dataset into train and test sets in 80:20.

X_train,X_test,y_train,y_test=train_test_split(iris.data,iris.target,test_size=0.20,train_size=0.80,random_state=42)

 # further splitting the train set into train and validate set.

X_train1,X_val,y_train1,y_val=train_test_split(X_train,y_train,test_size=0.25,random_state=42)

 # for k-fold cross validation. Here, CV=10 does 10 fold splitting performing test on 1 fold in each split.

cross_val_score(estimator, X_train, y_train, CV=10)

Citations:

- https://machinelearningmastery.com/data-leakage-machine-learning/

Usage a validation test.

1. **In short, if we do this directly on test data it will create a bias towards the test data.**

Detail:

2. We have to find the best values of the hyper parameters for the classifier to develop a robust model. Thus, we have used the validation dataset to experiment with different values of hyperparameters like n_neighbors and other parameters (if we wish) to find the hyperparameter set giving the best accuracy.
3. Then we use those hyperparameters to make the final predictions on the unknown set of completely isolated test dataset **for which the model is carrying no bias.**

What If:

- If we directly evaluate on the test set while finding the best parameters and then test the same data tuning with the best parameter found during the previous iteration, we would simply overestimate the accuracy as there is a bias towards the test data for that value of k.
- Because, the test data is already exposed to the model for finding the best hyperparameter, so taking that hyperparameter and testing the same data will overestimate the accuracy.

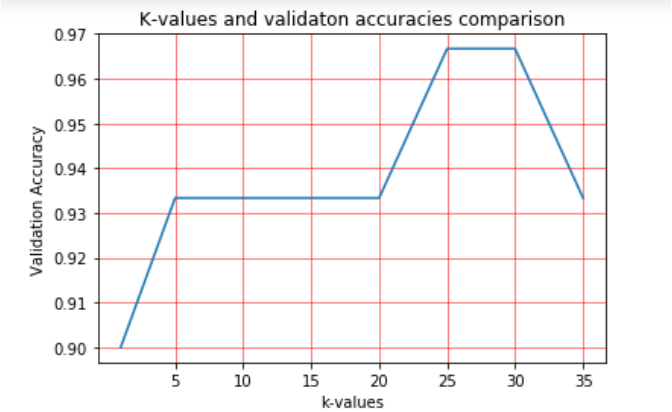
So, we evaluate the performance of our model by trying different combination of hyperparameters on validate set and selecting the one giving highest accuracy. At the end, the test dataset can be used to compare various models as that data was not used in either training or validating dataset.

Citations:

- https://ai.stanford.edu/~ronnyk/glossary.html
- https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets

The effect of changing k for KNN.

The graph of accuracy with corresponding value of k varies depending on the dataset. In this dataset, when we increase the value of K while testing the validation set, we have found that overall, the accuracy increases with the increase in the value of k to some extent but it is constant between k= [5,20] at 0.933. Further there is a gradual increase in the accuracy from 0.933 to 0.966 at k= 25, 30 with a further decline to 0.933 at k=35.



Reason: So, it can be seen that accuracy is not affected the same way with an increase in the value of k.

1. **When the value of K is 1**, model is only looking for **1 neighbor** and the mean accuracy is low as the model is making wrong predictions at some data points because of **overfitting and high variance**. Also when there is some new test data, model will not be able to correctly predict the output with a smaller k. That is, model is not generalizing well on the data it has not seen before.
2. **Between k= [5, 20]**, the accuracy has increased to 0.933 as the model is incorporating more neighbors to classify which is resulting in low variance. If we further increase the **k to 25, 30**, the accuracy increases to **0.966** where the variance is low as well. The bias or training error may be high at such value of k.
3. When the value of **k is high i.e. k=35**, model is making predictions by looking at 35 neighbors which has increased the computational complexity tremendously and the match with the true boundary line is very low and as a result it is giving us an accuracy of 0.933.

Note: We can find the variance by leave one out method in order to verify the results or by using the practical estimate of variance.

Citations:

- http://scott.fortmann-roe.com/docs/BiasVariance.html

Effect of changing the max depths for decision tree and random forests.

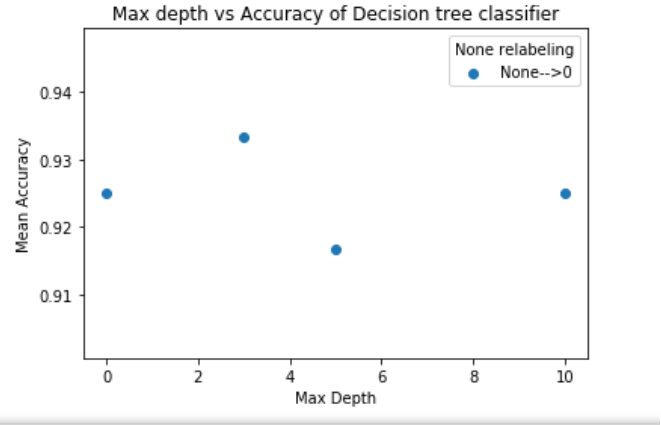
Maximum depth basically determines the depth up to which the nodes of a tree are expanded. ‘None’ in the question determines that the nodes should be expanded until the minimum numbers of samples are required to split the internal node.

DECISION TREE

Input Parameter: *max_depth= [3, 5, 10, None]*

Mean accuracies at multiple values of depth.

Mean Accuracy: 0.9333333333333332 at depth= 3
Mean Accuracy: 0.9166666666666666 at depth= 5
Mean Accuracy: 0.9249999999999998 at depth= 10
Mean Accuracy: 0.9249999999999998 at depth= 'None'



OBSERVATIONS AND ANALYSIS:

1. **High Depth:** It can be clearly seen that with the increase in the maximum depth at [10,None], the accuracy is little low as compared to the one with less depth because as the tree grows taking multiple decision functions into consideration, **it tends to over-fit the training data** and the accuracy for training data may be 100%. As a result**, it misses the important decision patterns** that it had to learn considering an appropriate level of depth.
   So when the new and novel data with unseen attribute values is fed as the validation/test data into the model, it is not able to predict the correct target class because the model lacks the learning capabilities for certain attributes at certain levels.

Note: This is not happening too much in our case which can be clearly seen as the less decline in accuracy at higher depths when tested on validation set as the model is generalizing the unknown data well even having higher depths because of less variable features in the dataset. But the more deep we are going, the complexity of the model is increasing.

Example of overfitting: At a particular level of depth, Let's say

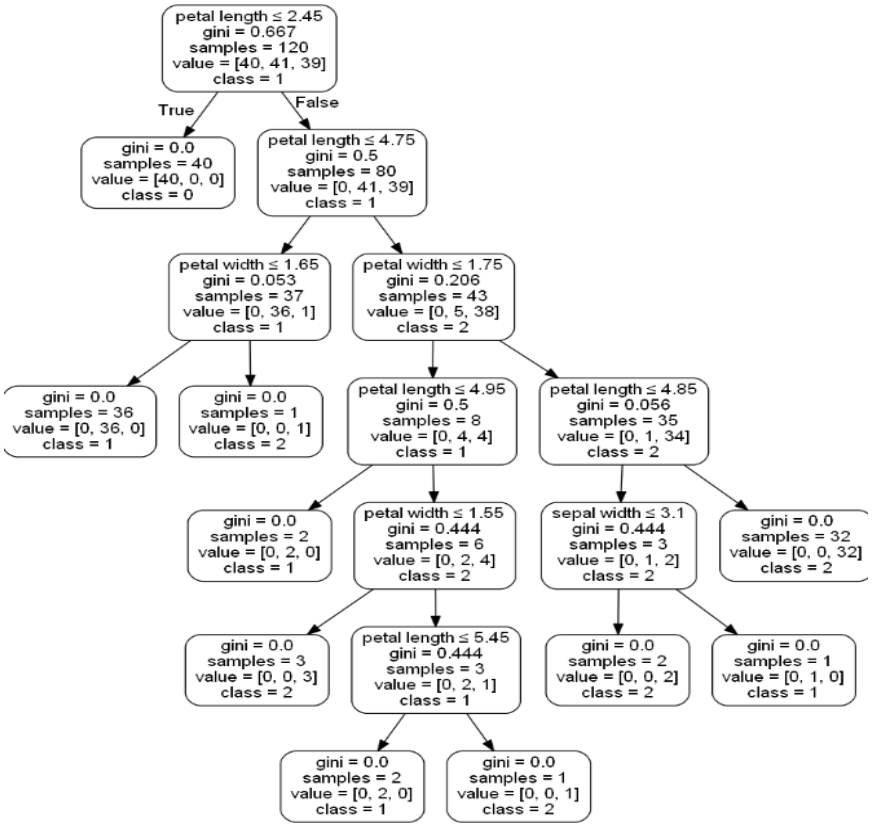
Decision functions

- sepal length>5.0: If true, petal length>2.0 : If true, petal width> 1.0: if true: sepal length>6.0 if true: petal length>4.0 if true: sepal width>2.0 →class 2.
  This is a case of over fitting.

If a new testing data is having:

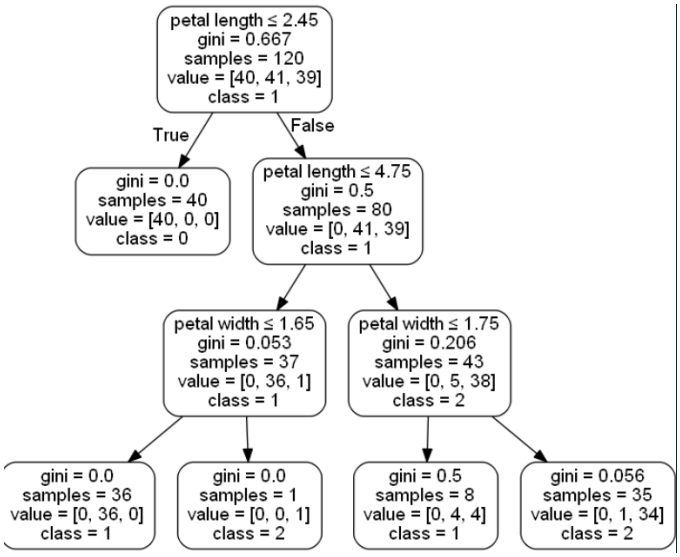
Sepal length =7.0, petal length=4.7, petal width=1.4 and sepal width=3.2

It will classify it wrongly as class 2 but it is class 1 and the accuracy decreases.



DEPTH=10

2. **BEST DEPTH= 3** is giving us the highest accuracy which is ***0.93*** in our case because it is taking two features **(Petal length and petal width)** to generalize the data which are separating all the three classes giving highest accuracy which can be seen using **graphviz. There is no over-fitting and no under-fitting at this depth.**



DEPTH=3

3. Generally, **when the depth is too low**, model again misses to learn the patterns because of less exposure to different decision functions covering different features, so in the end it also misclassifies the data points for both the training and the testing data as a result of under-fitting. This can be seen if we pass depth as **[1, 2]** in our classifier, the accuracy decreases to around **69% and 89% respectively**.

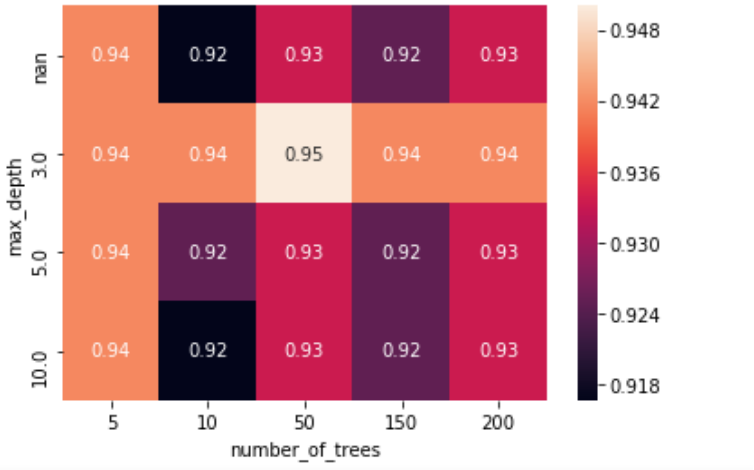
FOR RANDOM FOREST

number_of_trees	5	10	50	150	200
max_depth					
NaN	0.941667	0.916667	0.933333	0.925000	0.933333
3.0	0.941667	0.941667	0.950000	0.941667	0.941667
5.0	0.941667	0.925000	0.933333	0.925000	0.933333
10.0	0.941667	0.916667	0.933333	0.925000	0.933333

OBSERVATIONS AND ANALYSIS:

1. **We get the highest accuracy of 95% for our model at depth= [3] and at number of trees= [50].**

   a. **Depth =3** gives the same accuracy of 94% at different number of trees except when trees are 50 which is the optimum value for the classifier here and gives us the accuracy of 95%.
   b. **No of trees=50** is equalizing if any bias is created which is not created at this depth and meanwhile **covering best features i.e. Petal width and petal length which alone can separate most of the data with respect to 3 classes.**
   c. As the number of trees increases to 50, our model uses more random samples to build trees and when the depth is 3, it is using best features to generalize the data (petal width and petal length).



2. **When the number of trees= [150,200] and depth= [5, 10, None], the accuracy stands at 92.5% and 93.3%.**

    a. In decision tree, as we tend to increase the depth, the accuracy for test data also starts to decrease because it becomes the case of overfitting. Here, in ***random forest*** when we are increasing the depth, **it is also making multiple decision trees taking different samples as per n_estimators.**
    b. So the prediction from the multiple decision trees on voting helps to resolve the problem of bias created with higher depth. Hence, the accuracy is not dropping to that extent when we are increasing the depth **but the time complexity is increasing tremendously.**

Note: **There is no overfitting in the case of random forests because it equalizes all the biases and contradictions created as it follows the procedure of voting from multiple decision trees for selecting the final output class by either classification or regression.**

Citations:

- https://res.cloudinary.com/dyd911kmh/image/upload/f_auto,q_auto:best/v1545933329/output_57_0_livyu1.png
- https://en.wikipedia.org/wiki/Decision_tree_learning
- https://www.datacamp.com/community/tutorials/decision-tree-classification-python
- https://www.datacamp.com/community/tutorials/random-forests-classifier-python

Effect of the number of estimators for Gradient Tree Boosting.

Mean accuracy and no of estimators :

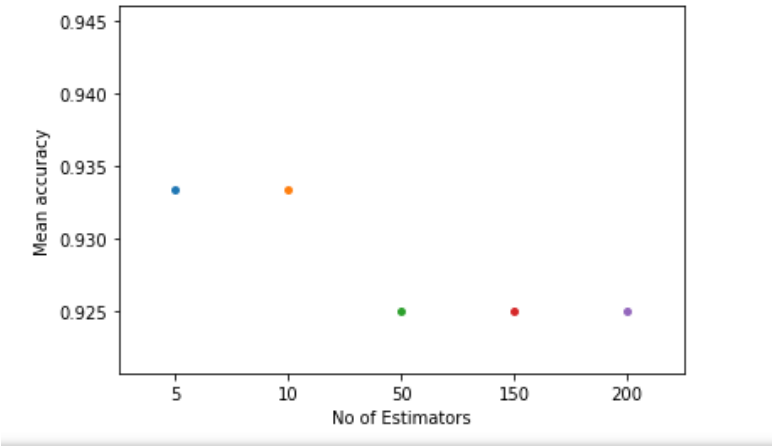
	No of Estimators	Mean accuracy
0	5	0.933333
1	10	0.933333
2	50	0.925000
3	150	0.925000
4	200	0.925000

OBSERVATIONS AND ANALYSIS:

1. **At no. of estimators= [5, 10]:**

    a. It can be seen from the table, that the best mean accuracy i.e. 93% is coming when the number of estimators= [5, 10] which are kind of the best values in our case because there is no overfitting happening at this point and it is enough to clear the errors made by the predecessor models.

b. Thus, Number of estimators= [5, 10] are good to learn the patterns about the data for our model. So when the model is tested with new data, it will generalize well and giving us descent accuracy.



At no. of estimators= [50,150,200]:

1. The accuracy has decreased to 92.5% at no. of estimators= [50,150,200].
2. So, these values are giving higher accuracy when tested on the training data but when unseen data is fed into the model, it is not generalizing well.
3. The time complexity for training the model has also increased drastically.

Accuracy comparison between Gradient boosting and Random forest:

Best Accuracy:

1. **Accuracy of random forest** is coming 95% at depth= [3] and number of trees= [50] because depth=3 is enough to **learn the patterns in the data** and no of trees=50 **are good enough to clear the bias.**

2. Whereas in **gradient boosting**, the best accuracy is 93% at n_estimators= [5, 10] which is less than that of random forest because we have not tuned the parameters of GB classifier and it doesn't perform generally well in the presence of noise. (It is not there in our dataset).

Note: In our case, the gradient boosting can generalize well if we tune the hyper parameters well.

Citations:

- https://medium.com/all-things-ai/in-depth-parameter-tuning-for-gradient-boosting-3363992e9bae
- https://www.dasca.org/world-of-big-data/article/gradient-boosting-for-beginners?utm_campaign=News&utm_medium=Community&utm_source=DataCamp.com

Parameter C and its effect in SVM classifier.

C is basically a regularization parameter whose functionality is used to adjust the misclassified data points. It is the controlling parameter for the adjustment between decision boundary and the classifying training points correctly.

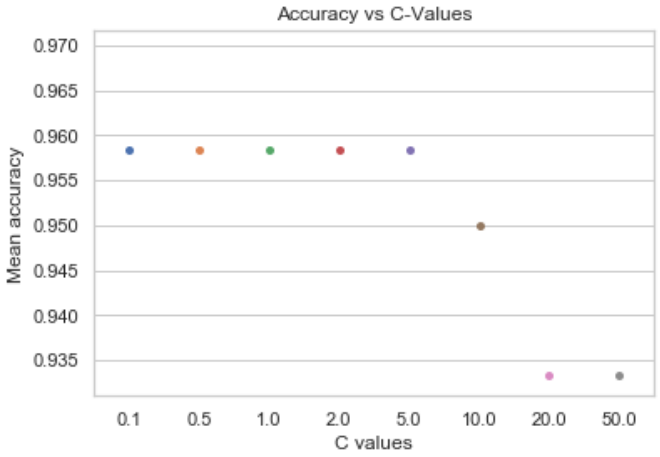
1. For the low values of C, the margins are higher with little or no misclassification for the training data and low accuracy for test data and vice versa.

2. There is no hard rule to find the optimal c value but we can use different values in validation set to select the one giving maximum accuracy or by using gridsearchCV.

The error is basically given as

Error= C * classification Error + Marginal error

So small C will give large margin with some classification errors. A large C will give smaller margin with no/lesser classification errors for training data depending on the data.



1. When the value of C is given as **[0.1, 0.5, 1,2]**, the margins are higher and there is some bias and misclassifications on the training data but the overall accuracy on the validation set is higher.

2. When the c is high i.e. at c= **[20, 50],** the margins are low but for the new test/validation data, the accuracy is low.

3. We need to choose a c which gives higher margin with no or less misclassifications. The value of C depends on the type of future data.

4. A low c value is best if it misclassifies 1 point rather than a high C misclassifying more points with low margins.

5. On choosing the best C= [0.1,2] from the validation set, when we use it on test set we get the accuracy of 1.0 i.e. with higher margins these values are not making any misclassification errors.

6. when the value of C= 0.1, the margins will be high but there will be some sort of misclassification for training data which may not be reflected in predicting the test data and this may also give high accuracy for test data which can be seen when passed for test data. Thus, it misclassifies the linear training data too.

Citations:

- https://medium.com/@pushkarmandot/what-is-the-significance-of-c-value-in-support-vector-machine-28224e852c5a
- https://medium.com/towards-artificial-intelligence/support-vector-machine-svm-a-visual-simple-explanation-part-1-a7efa96444f2
- https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel