INTRODUCTION TO ARTIFICIAL INTELLIGENCE

COMP6721

Mini Project 2 Report

Character Recognition

Submitted By – Amandeep Singh (40052070)

Contents

Basic Experimental Setup 3
Experimentation & Analysis 7
DATASET 1 7
DATASET 2 12
Conclusion 16
References 17

Basic Experimental Setup

This project focuses on Machine learning algorithms, to train a system for character recognition. The characters include 26 English alphabets (Dataset1) and 10 Greek alphabets (Dataset2). We train the system for around 1,950 instances for Dataset1 and around 6,400 instances of Dataset2. The trained models are then tested for new, unclassified characters. It's a classification problem wherein we first train the system, adjust hyperparameters for increasing the system accuracy using the validation data, save the trained model and then use this saved model to classify new, unclassified characters.

I'm using Python (scikit-learn library) to train the system for the following Machine Algorithms:

1. Decision Tree
2. Naïve Bayes Classifier
3. Support Vector Machine

1. Decision Tree

   This algorithm is explained in the class. "The scikit-learn uses an optimised version of the CART algorithm" [1]. The CART (Classification and Regression Trees) algorithm is the successor for C4.5 algorithm. The following hyper-parameters were used to improve the accuracy for this model:

   a) max_depth

      It's a kind of pruning factor for a decision tree. It indicates how deep the tree can grow. A deeper tree has more splits and captures more information about the data but takes more time to train and possible chances of producing an overfit model. For this project, I experimented with depths ranging from 1 to 60, while the default value is None for this parameter.

   b) min_samples_split

      This parameter indicates how to split an internal node based on the number of samples required. It varies from considering just 1 sample at each node to considering all the samples. The more samples we choose, the more the tree becomes constrained and thus takes more time for computations. For this project, experimented with this value ranging from 2 to 200, while the default split requires 2 minimum samples.

   c) min_sample_leaf

      It's similar to the min_samples_split parameter but describes the minimum number of samples required at the leaf node. A split point at any depth is only made if the resulting left and right trees have talent min_sample_leaf training samples. In the project, I experimented its values from 1 to 100, while the default value is 1.

   d) max_features

      It represents the number of features to consider when looking for a best split. While it default value is None, for the purpose of experimentation, I varied it from 1 to 100.

2. Naïve Bayes Classifier

   This algorithm is also explained in the class. It's based on the Bayes Theorem for conditional probability which the assumption of conditional independence of features. The following hyper-parameters were used for accuracy improvement:

   a) Gaussian Naïve Bayes

      It assumes the Gaussian distribution of the features in training data characterized by mean and variance.

   b) Bernoulli Naïve Bayes

      It assumes the binary distribution of the features i.e. a feature can be present or absent. Since a feature in our training data represents if it is on or off for a particular character, i.e. the binary distribution of features. So, this classifier should perform better for our data.

   c) Multinomial Naïve Bayes

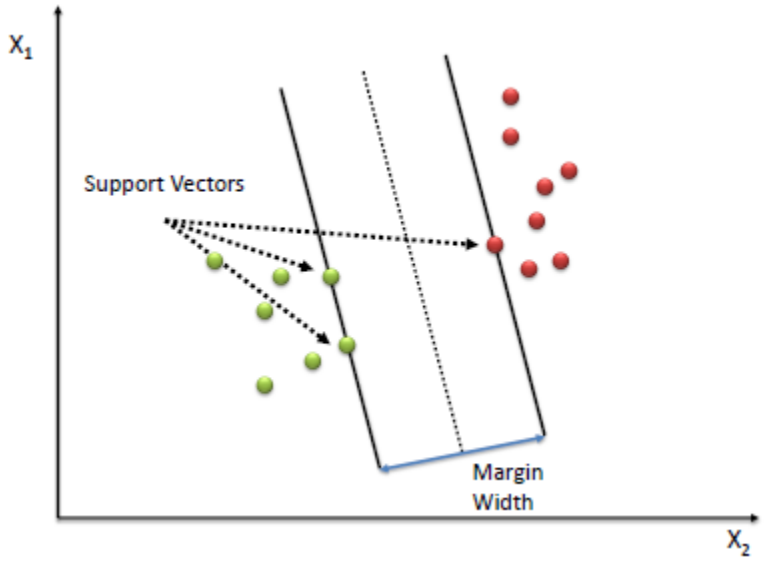
      It assumes the discrete distribution of the features. i.e. a feature is represented by a whole number.

   d) alpha

      It's a smoothing parameter used in Bernoulli and Multinomial Naïve Bayes classifiers. Its idea is to increase the conditional probability of features to non-zero value. In the project, I experimented its value from 0.1 to 1, while the default value = 0.

3. Support Vector Machine, SVM

   It's a supervised machine learning model that analyze the data for classification, regression and outlier detection. This algorithm tries to find those training samples which maximizes the gap between different classes of data in the sample. These training samples are called the Support Vectors and once these vectors are found, the classification of new points into different classes can be done just with the help of these vectors.



Support Vector Machine [2]

I choose this algorithm because of its following advantages [3]:

1. It's very effective in high dimensional spaces i.e. when we have lots of features in our training data. In the project, there are 1024 feature for each character.
2. It's effective when the number of features is more than the training data or at least comparable. In our project, for English character recognition, we have 1024 features and just around 1,950 training samples. So, the SVM can really perform better than the Decision Tree and Naïve Bayes classifiers.
3. It only uses support vectors to make decision for new data, so its memory efficient.
4. It's versatile in the sense, that it provides different kernel functions for making decisions. It also provides flexibility of defining custom kernel functions.

The following hyperparameters were used for improving the model accuracy:

a) kernel
   As stated above in the advantages, the SVM provides different kernels for making the decision. I performed experimentation using these kernels:
   i. linear
   ii. rbf (used for non-linear hyper plane classification)
   iii. sigmoid
   iv. poly (used for non-linear hyper plane classification)

b) C
   It's a penalty parameter for the error term. It's a trade-off between smooth decision boundaries and proper classification of training samples. I varied its value from 0.001 to 100 for experimentation, while the default being 1.0

c) gamma
   It's the kernel coefficient for 'rbf','poly' and 'sigmoid' kernels. High values of gamma try to exact fit the training data thus causes overfitting problems. I experimented its values ranging from 0.001 to 100. Its default value is 'auto'.

d) coef0
   It's a parameter for kernel projection in 'poly' and 'sigmoid' kernels. It's a smooting parameter similar to the alpha parameter of Naïve Bayes classifier. I experimented it's values in the range of 0.001 to 100, while it defaults to 0.0 in the scikit library.

Additional Code for Hyper-parameter testing.

As for the SVM, I'm testing for 4 hyper-parameters, so I needed something which can exhaustively perform training for all possible combination of these parameters. In total, there were 1,309 possible combinations of these 4 hyper-parameter values. Performing this manually is a cumbersome and error prone task. So, I used a handy feature of ***ParameterGrid*** provide by scikit-learn library. It generates all possible combination of the parameters which we can use by iterating over them.

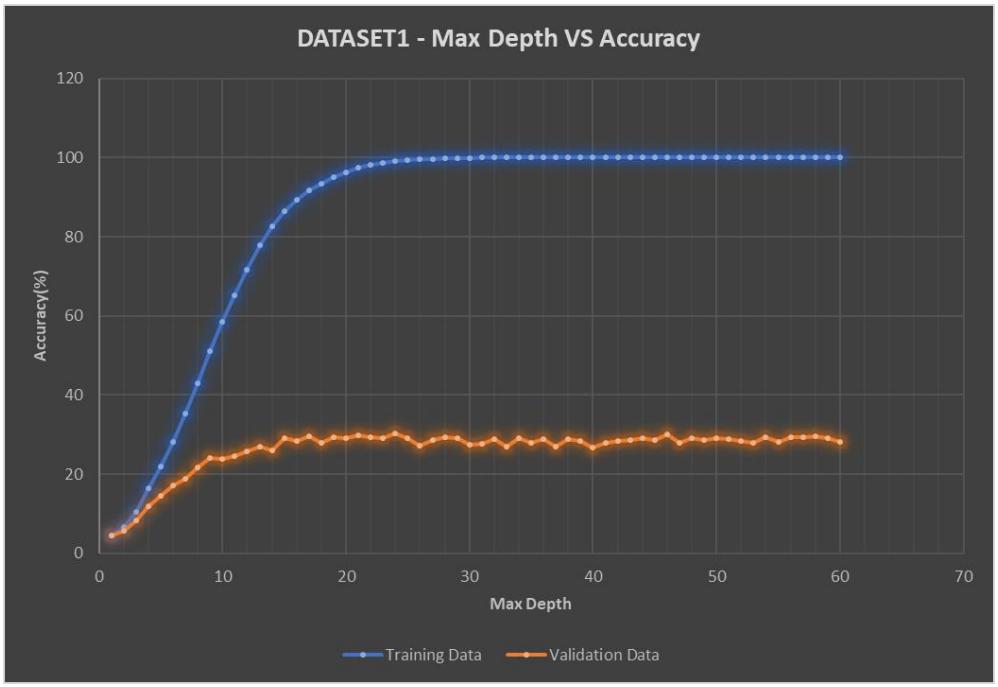
```
grid = [{'kernel': ['rbf'],
         'gamma': np.logspace(-3, 2, 6),
         'C': [1e-3, 1e-2, 1e-1, 1, 10, 100, 1000]},
        {'kernel': ['sigmoid'],
         'gamma': np.logspace(-3, 2, 6),
         'coef0': np.logspace(-3, 2, 6),
         'C': [1e-3, 1e-2, 1e-1, 1, 10, 100, 1000]},
        {'kernel': ['poly'],
         'degree': [1, 2, 3, 4],
         'gamma': np.logspace(-3, 2, 6),
         'C': [1e-3, 1e-2, 1e-1, 1, 10, 100, 1000],
         'coef0': np.logspace(-3, 2, 6)},
        {'kernel': ['linear'],
         'C': [1e-3, 1e-2, 1e-1, 1, 10, 100, 1000]}]

paramGrid = ParameterGrid(grid)
```

Parameter Grid

Experimentation & Analysis

DATASET 1

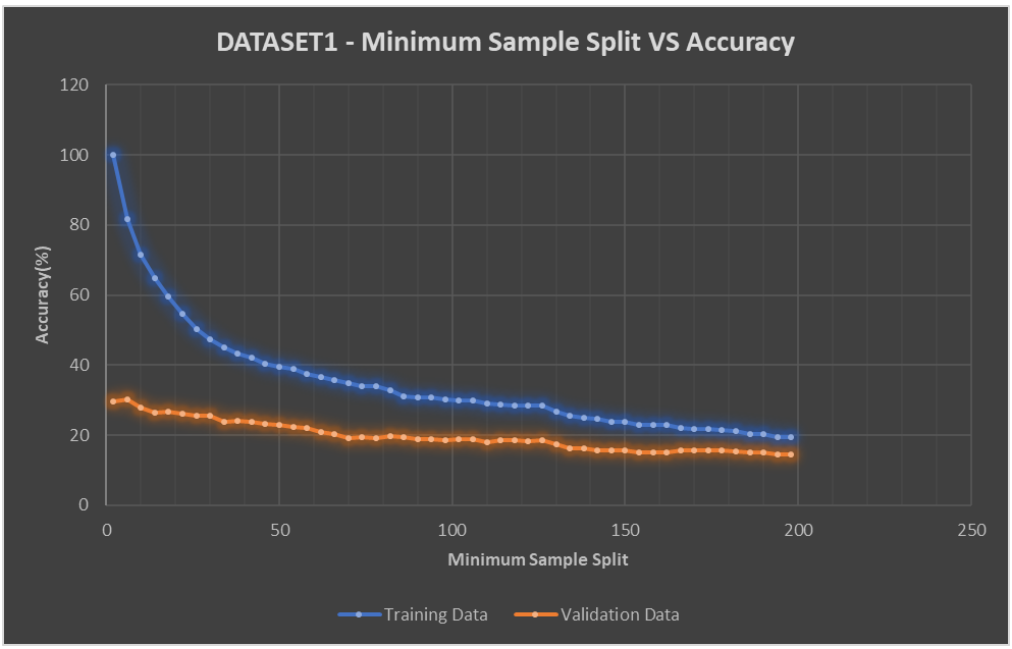
1. Decision Trees
   Attached Experimentation File: DecisionTree_Experiments_dataSet1.xlsx



ANALYSIS: We can see clearly, our model **over fits** for large depth values.

Values Analyzed: 60 (1 to 60)

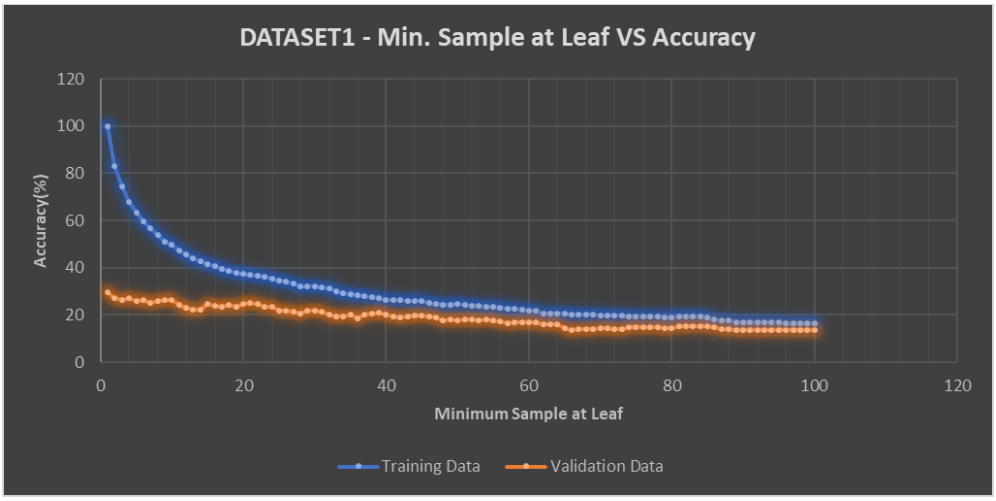
Max Accuracy for Validation Data: **30.35019455** at depth: **24**



ANALYSIS: Our model **underfits** for larger values of min. sample split.

Values Analyzed: 50 (2 to 198 with 4 increments)

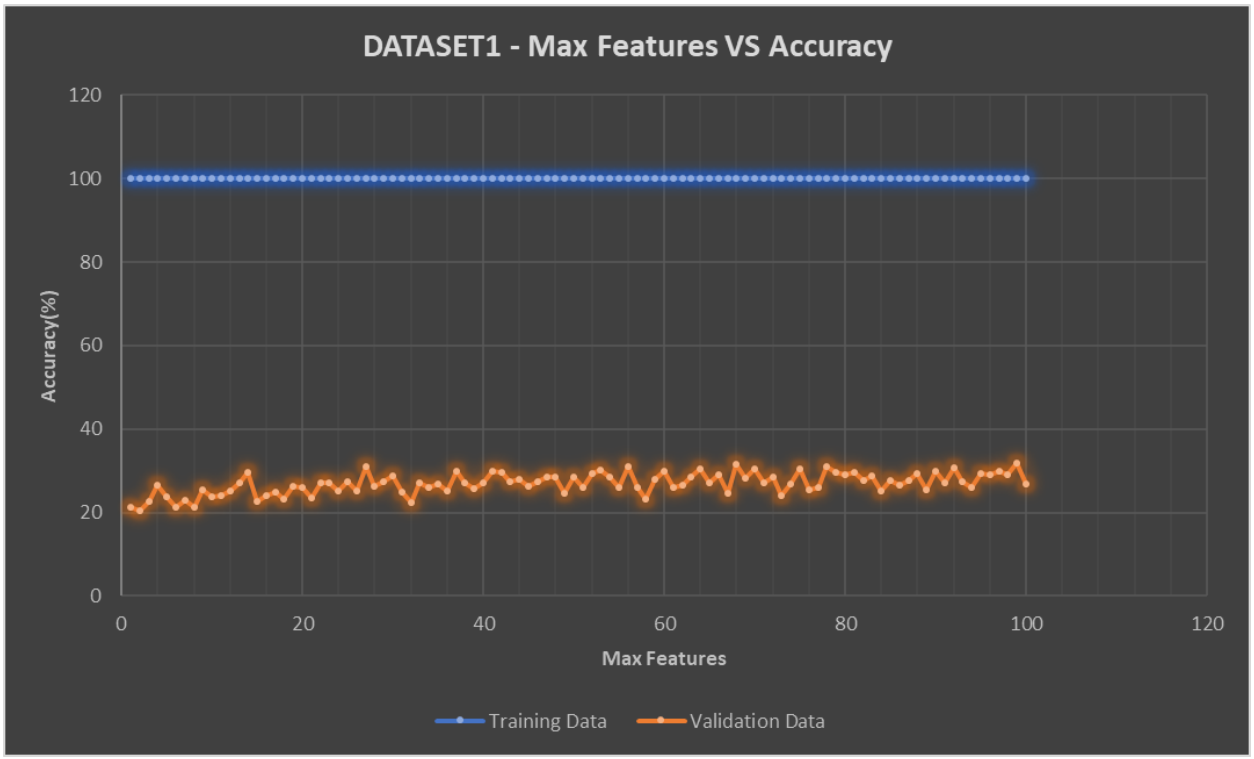
Max Accuracy for Validation Data: **30.15564202** at Min Sample Split: **6**



ANALYSIS: Similar to the previous one, our model **underfits** for larger values of min. sample leaf.

Values Analyzed: 100 (1 to 100)

Max Accuracy for Validation Data: **29.57198444** at Min Sample Leaf: **1**



ANALYSIS: It's an overfitting case. It's expected to get overfitting for all values of max_features

Values Analyzed: 100(1 to 100)

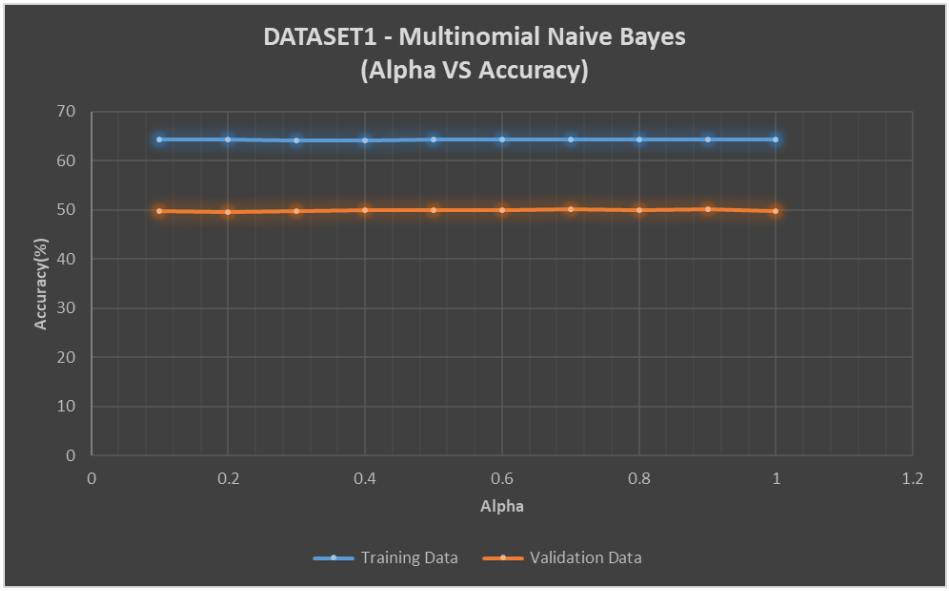
Max Accuracy for Validation Data: **31.71206226** at Min Sample Split: **99**

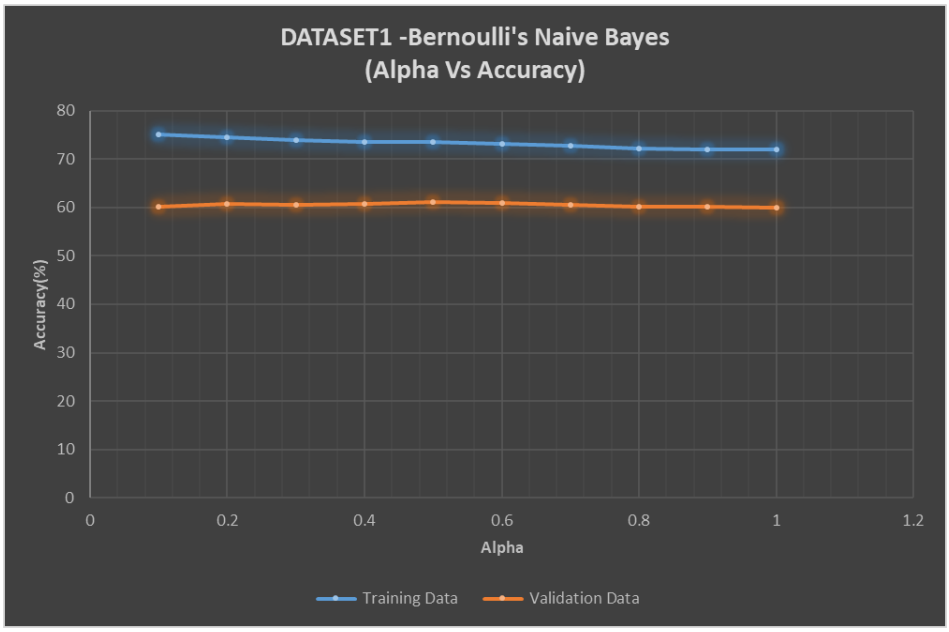
On combining all the hyperparameter values

Max Depth	Min. Sample Split	Min. Sample Leaf	Max Features	**Accuracy**	
24	6	1	Default	**28.79**	With configuration
Default	Default	Default	Default	27.23	Without config.

With hyper-parameter configuration, we gained the accuracy by = 28.79-27.23 = **1.56%**

2. Naïve Bayes Classifier
   Attached Experimentation File: NaiveBayes_Experiments_dataSet1.xlsx





Type of Naïve Bayes Classifier	Training Data Accuracy	Validation Data Accuracy
Gaussian	71.89	43.77
Bernoulli	71.10	61.09 (at alpha=0.5)
Multinomial	64.28	50.19 (at alpha=0.6)
Default Configuration	-	59.92

ANALYSIS: The maximum accuracy is given by Bernoulli Naïve Bayes for smooting factor(alpha) = 0.5.

It was expected since our dataset had binary distribution.

Values Analyzed: Bernoulli (10) + Multinomial (10) + Gaussian (1) = 21

Accuracy gained by hyper parameter configuration: 61.09 - 59.92 = 1.17%

3. SVM

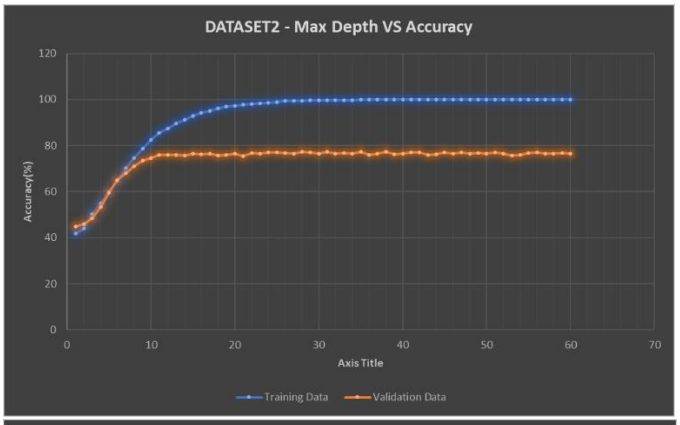
   Attached Experimentation File: SVM_Experiments_dataSet1.xlsx

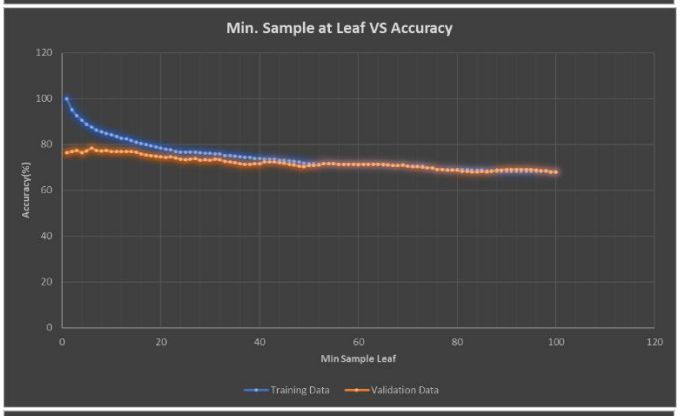
kernel	Parameters	Validation Data Accuracy	Values Analyzed
Linear	C=0.01	66.15	7
rbf	C=10, gamma = 0.001	66.93	42
Sigmoid	C=10, coef0=0.001, gamma=0.001	63.03	252
Ploy	C=1, coeff0=0.1, gamma=10, degree=4	69.65	833

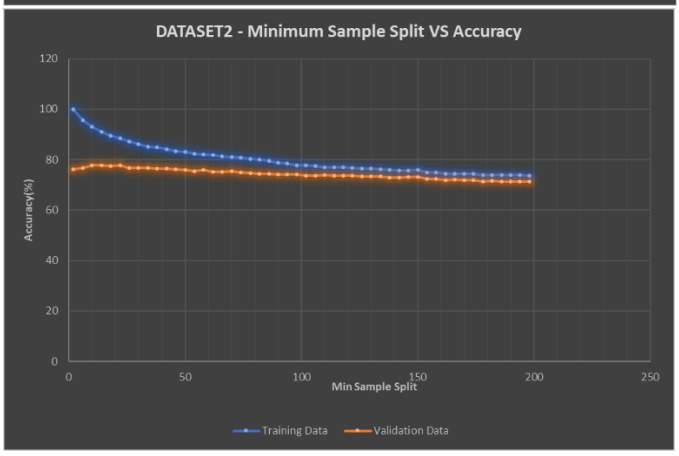
ANALYSIS: Max. Accuracy of 69.65 is given by **'ploy'** kernel with hyper parameter at (C=1, coeff0=0.1, gamma=10, degree=4)

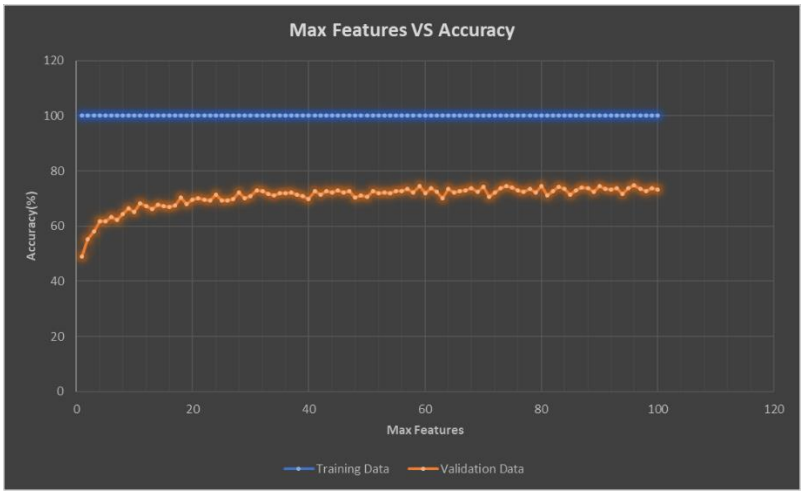
DATASET 2

1. Decision Trees
   Attached Experimentation File: DecisionTree_Experiments_dataSet2.xlsx









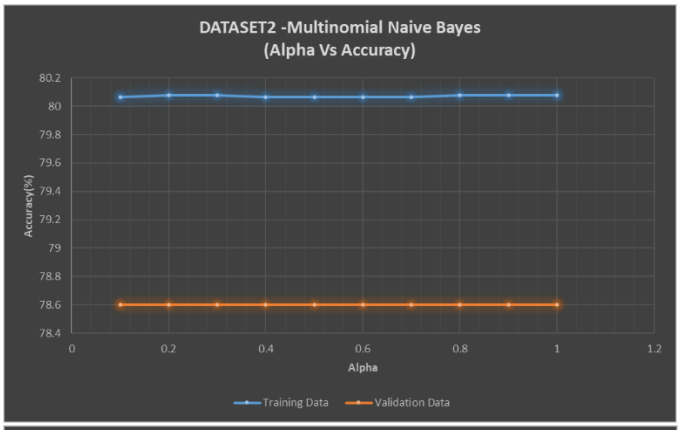
Parameter	Validation Data Accuracy (%)	Values Analyzed
Max Depth (=38)	77.4	60 (1 to 60)
Min Sample Split (=14)	77.7	50 (2 to 198 with 4 increment)
Min. Sample Leaf (=6)	78.15	100 (1 to 100)
Max Features (=96)	74.7	100 (1 to 100)

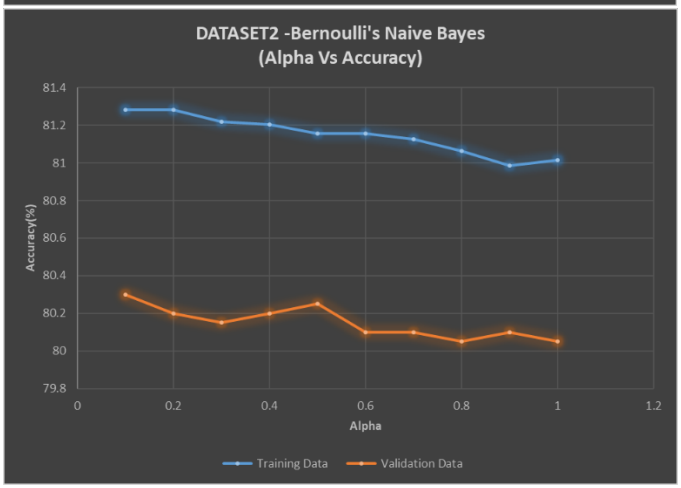
Combined Accuracy (max depth=38, Min sample split = 14, min sample leaf = 6, max features=96) = 71.95

Accuracy with default Parameters= 77.1

So, the combination of hyper parameters decreases the accuracy, for this I'm only using min. sample leaf hyper-parameter which gives the accuracy of 78.15

2. Naïve Bayes Classifier
   Attached Experimentation File: NaiveBayes_Experiments_dataSet2.xlsx





Type of Naïve Bayes Classifier	Training Data Accuracy	Validation Data Accuracy
Gaussian	64.29	64.2
Bernoulli	81.28	80.30 (at alpha=0.1)
Multinomial	80.06	78.60 (at alpha=0.1)
Default Configuration	-	64.2

3. SVM

   Attached Experimentation File: SVM_Experiments_dataSet2.xlsx

kernel	Parameters	Validation Data Accuracy	Values Analyzed
Linear	C=0.01	88.75	7
rbf	C=10, gamma = 0.01	94.15	42
Sigmoid	C=10, coef0=0.001, gamma=0.001	88.25	252
Ploy	C=1, coeff0=0.1, gamma=10, degree=4	69.64	833

   For this dataset, the ‘rbf’ kernel gives the max. accuracy of 94.15%

Conclusion

For Naïve Bayes, the Bernoullis Naive Bayes runs the best as our data have the binary distribution.

References

1. Decision Tree Implementations: https://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart
2. SVM Image: https://www.saedsayad.com/images/SVM_2.png
3. SVM Advantages: https://scikit-learn.org/stable/modules/svm.html#support-vector-machines
4. Naïve Bayes Parameters: https://hub.packtpub.com/implementing-3-naive-bayes-classifiers-in-scikit-learn/