SMAI - Mini Project 2
Technical Report

20161159

Data

The aim of this project is to perform classification tasks on CIFAR-10 dataset. There are 6 data batches each having 10000 entries - 5 “train” batches and 1 “test” batch. The composition of each batch is as follows:

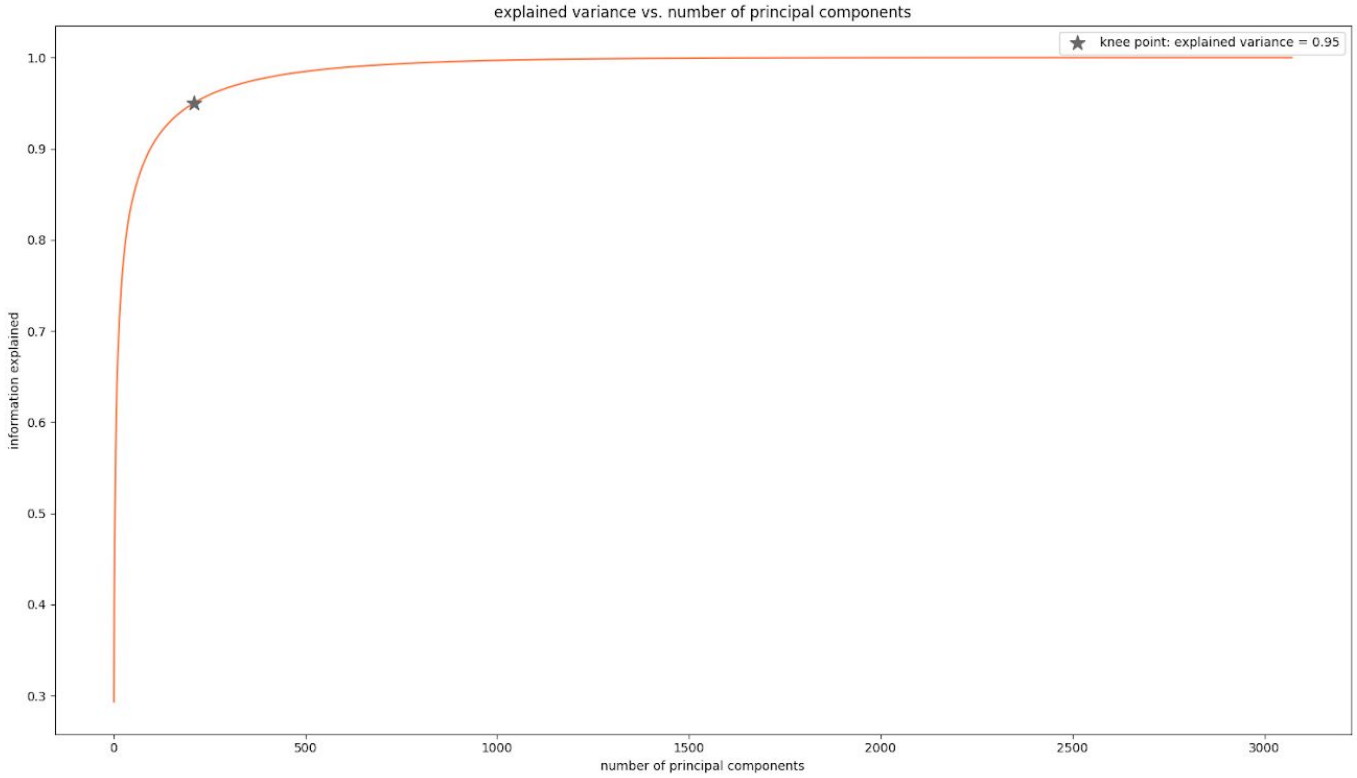
- **Data -** 10000x3072 numpy array of *unit8*s. Each row represents a 32x32 colored stored in row-major order.
- **Labels -** List of 10000 integers in the range [0-9] representing class labels.

This project requires us to perform many experiments of training models with different hyperparameters, and observing the results obtained. However for many classifiers like Linear SVM, training on even 1 complete data batch for each experiment takes too much time. Hence, I divided *data_batch_1* into sets of 1000 samples, trained the model on each set, and averaged the results obtained from each training set. Since the dataset is well shuffled, there is roughly equal representation from each class in the each of the training sets. Apart from this, the aim is not to get maximum accuracy, but to observe the change in results with varying hyperparameters. For each experiment with particular hyperparameter value, averaging the 10 results ensures that the result is not data-dependent and is actually explained by the particular hyperparameter.
Wherever training happens fast enough, I have incorporated the largest possible training set.

Data Representation

PCA

The sklearn.decomposition.PCA module allows us to specify *0 < n_components < 1* such that we can specify the amount of information we want to retain. I set *n_components = 0.95*, and this resulted in a dimensionality reduction from 3072 -> 209 dimensions. In the graph below, we can see that the plot of *explained variance* vs. *number of principal components* has a knee-point roughly at 209 principal components. Increasing the number of components past this knee-point leads to extremely low increase in retained information.



LDA

LDA (Linear Discriminant Analysis) fits a Gaussian density to each class. The data is then projected to the most discriminative directions, and hence dimensionality is reduced. LDA allows a maximum of *(n_classes - 1)* components, and hence for our dataset we can have at max. **9 components**. Reducing dimensions more than this doesn't give us much computational gain, whereas we lose out on a fraction of information. Hence, it is best to fix the number of components at 9.

Classification

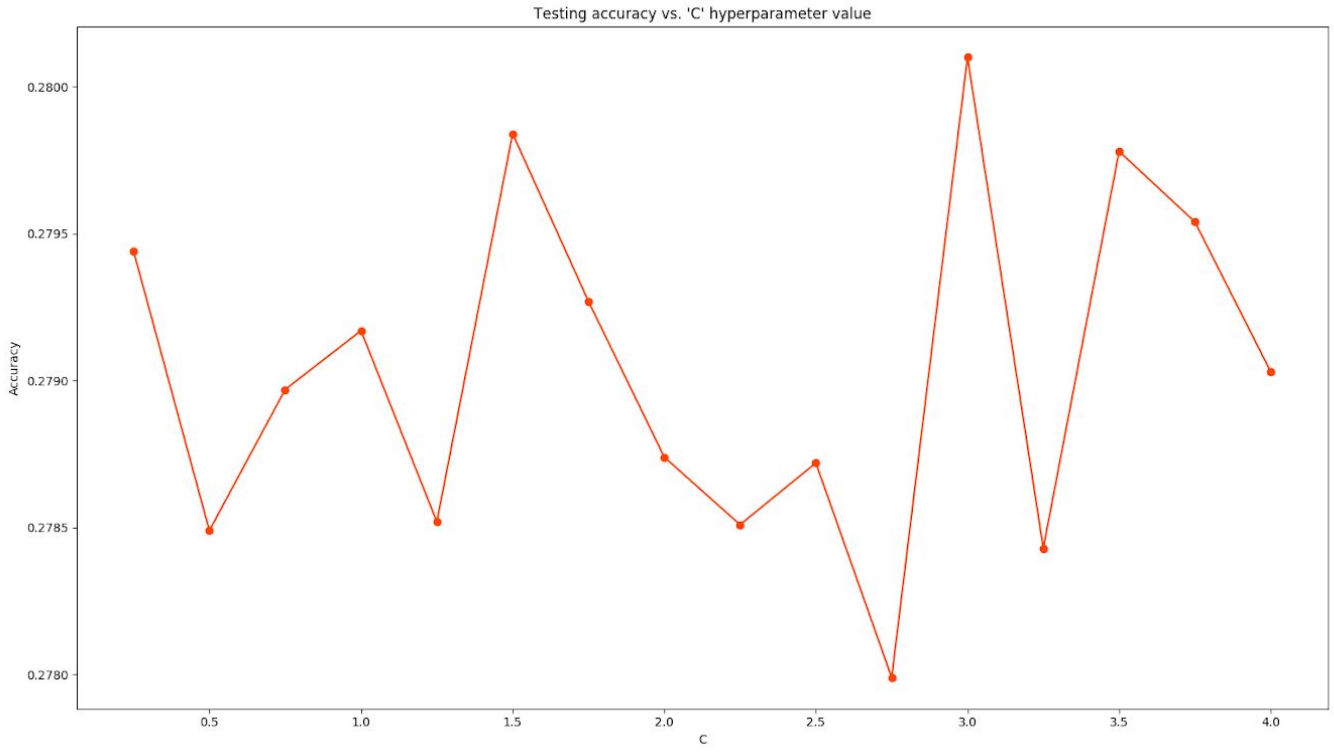
1) Soft-margin Linear SVM

Hyperparameter Description

In hard-margin linear SVM, the objective is to maximize the margin without allowing misclassifications. Hence, there would be no hyperparameter in this case. In soft-margin linear SVM, misclassifications are allowed. The objective is to maximize the margin while minimizing the misclassification error. The hyperparameter **C** is the "penalty term", and its value determines how much to penalize misclassification error.

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i$$
$$\text{s.t. } y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i, \forall i \in [l]$$
$$\text{and } \xi_i \geq 0, \forall i \in [l]$$

Hyperparameter Selection



We get the best accuracy with **C=3.0**.

Note on overfitting: We can see to the right side of the plot, as C takes higher values after 3.0, the testing accuracy is not so good anymore. We need C to be sufficiently large to penalize misclassifications so as to have a correct classificier. However C should not be too large, otherwise it will overfit the training data. In this case, higher priority will be given to making a correct classifier on the train data, whereas lower priority will be given to maximizing the margin.

Results Table

For Soft-margin Linear SVM classifier (with **C=3.0**) we get the following results for various data representations:

Representation	Accuracy	F1-score
Raw Data	0.29172	0.29172
PCA	0.28	0.28
LDA	0.24973	0.24973

2) Kernel SVM with RBF Kernel

Hyperparameter Description

In SVM with RBF Kernel, there are 2 hyperparameters - **γ** and **C**.

$$K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2), \gamma > 0$$

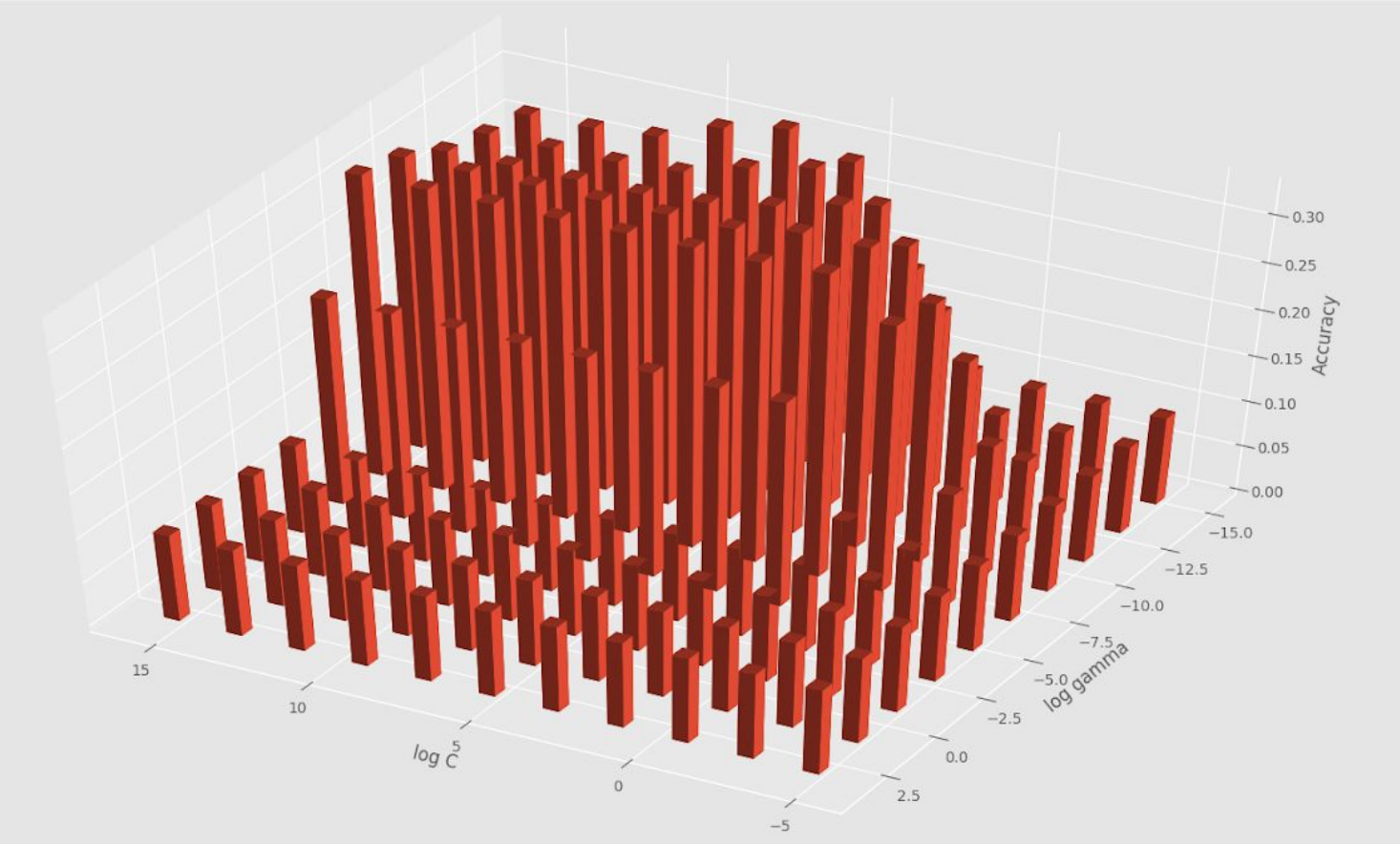
γ is the constant in the Kernel function. It's value indicates the inverse of the radius of influence of each sample which is selected as a support vector. **C** is the penalty for misclassification error in the objective function.

Hyperparameter Selection

Since there are 2 hyperparameters involved, we use **grid-search** rather than linear search to find the best pair of **(C,γ)**. I followed the following steps to find the best parameter pair:

1. Preprocess the data so that all features are scaled to the range [0,1]. If the input values are too large as compared to γ, the model will always return the same exact value.
2. Calculate the accuracies of models created from all pairs of C & γ, such that C belongs to $[2^{-5}, 2^{-3},.....,2^{13},2^{15}]$ and γ belongs to $[2^{-15}, 2^{-13},.....,2^{1},2^{3}]$.
3. Choose the (C,γ) which gives the highest accuracy.

In the image given below, we can see the accuracy of the model trained on all different (C,γ) hyperparameter pairs:



We find that the largest peak is at **log(C)=1** and **log(γ)=-7**.
Thus, we get maximum accuracy by selecting hyperparameters as **$C=2^{1}$** & **$\gamma=2^{-7}$**.

Results Table

Results achieved with RBF Kernel SVM with hyperparameters $C=2^{1}$ and $\gamma=2^{-7}$.

Representation	Accuracy	F1-score
Raw Data	0.3337	0.3337
PCA	0.3344	0.3344
LDA	0.1947	0.1947

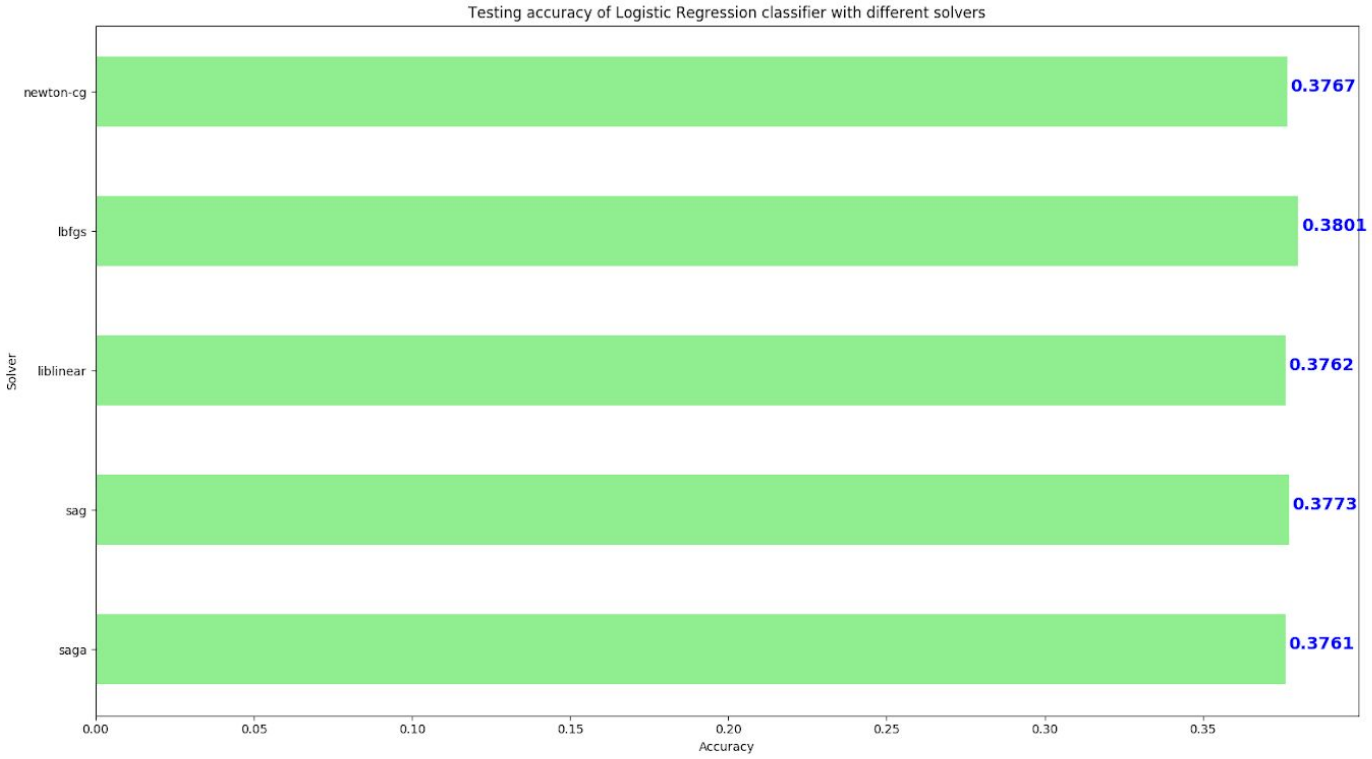
3) Logistic Regression

Hyperparameter Description

In Logistic Regression, the **solver** hyperparameter gives us a choice of 5 different algorithms to use in the optimization problem: {"newton-cg", "lbfgs", "liblinear", "sag", "saga"}.

Hyperparameter Selection

For each solver, I performed logistic regression on the entire training data batch (10000 samples) since it was fast enough.



- It can be seen that all solvers perform almost equally well.
- **Time taken to converge-** *newton-cg:* 24s, *lbfgs:* 15s, *liblinear:* 1m58s, *sag:* 12s, *saga:* 26s.
    - *liblinear* takes much more time as it uses the one-versus-rest scheme, whereas all the other solvers use multinomial loss.
    - When features have approximately same scale, *sag* and *saga* are guaranteed to converge fast. Since in this case each feature is a pixel taking value from [0-255], fast convergence is guaranteed.
- Since performance is roughly uniform and *sag* converges much faster, I have chosen *sag* for the **solver** hyperparameter.

Results Table

For Logistic Regression using *sag* solver, for each data representation the following results were obtained:

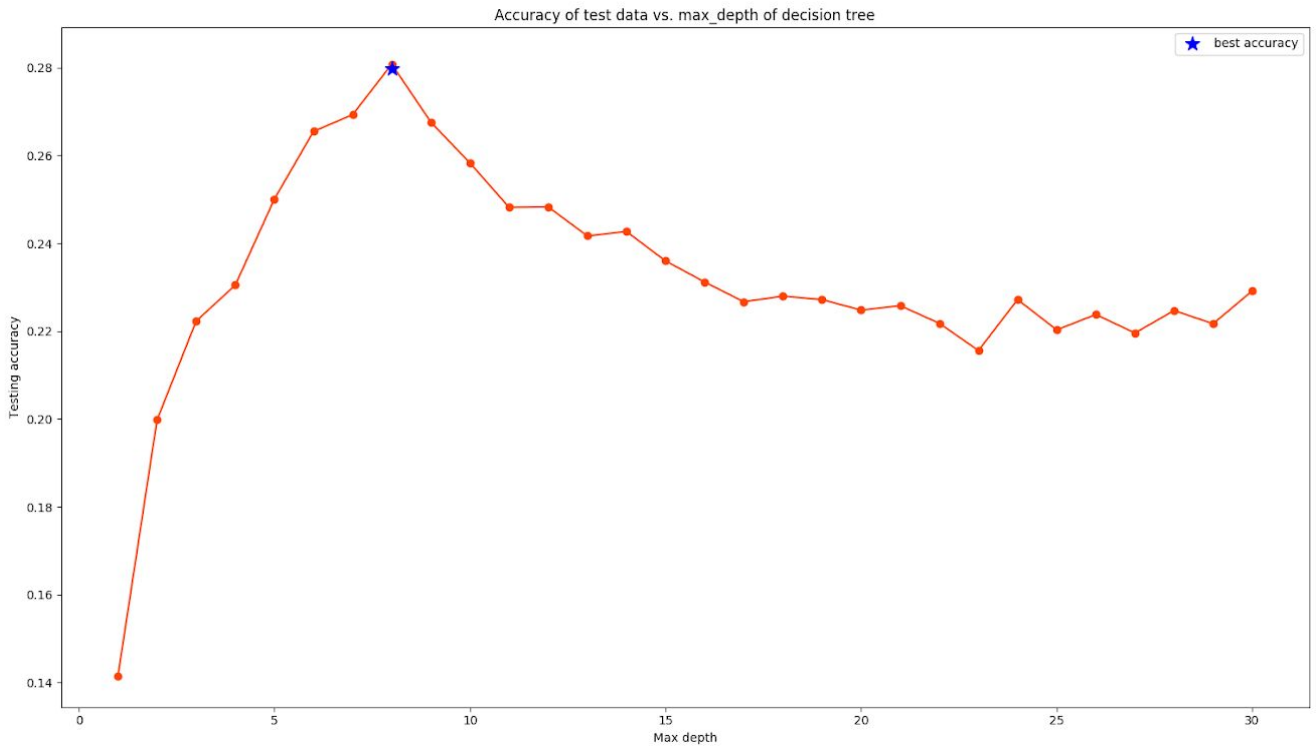
Representation	Accuracy	F1-score
Raw Data	0.401	0.401
PCA	0.3773	0.3773
LDA	0.259	0.259

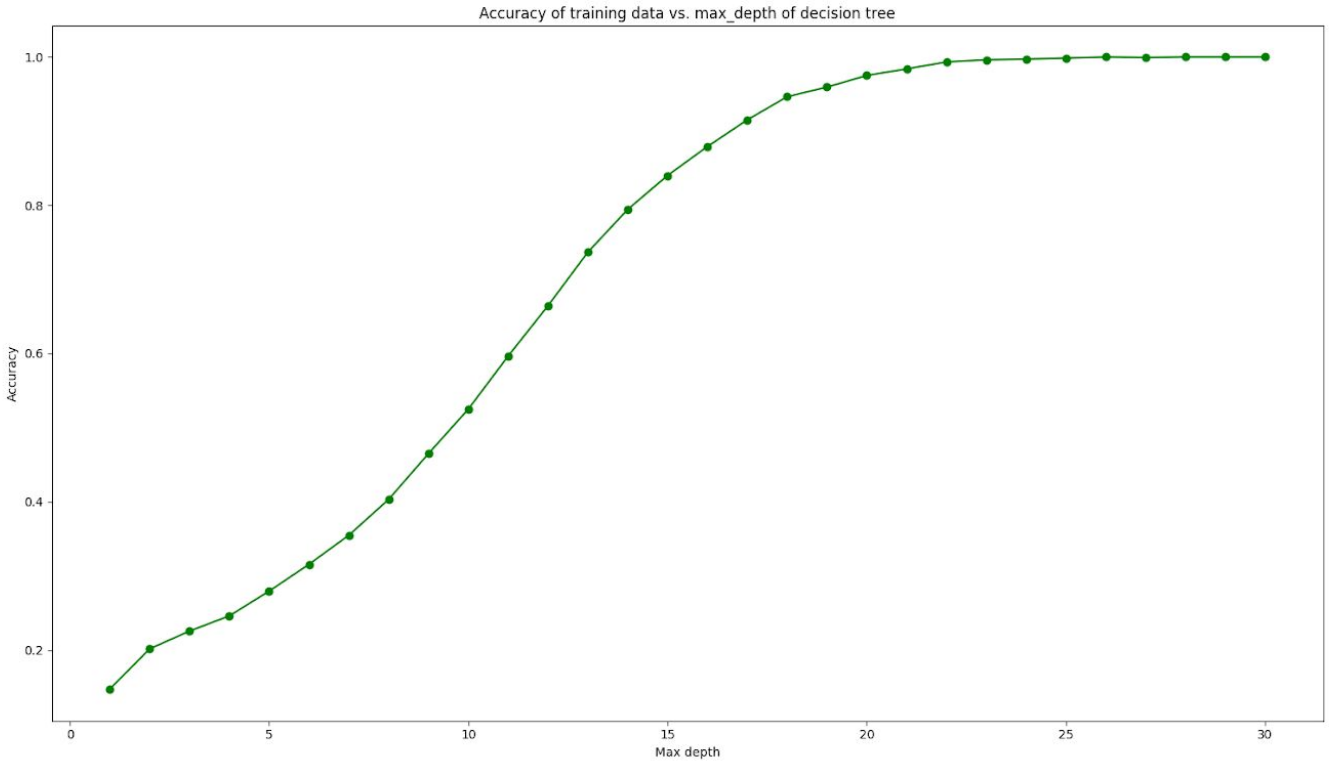
4) Decision Tree

Hyperparameter Description

I have considered one hyperparameter, **max_depth**. This hyperparameter limits the depth of the decision tree. Normally nodes are split until a leaf or pure node is reached. However, varying this hyperparameter can stop the splitting at an earlier level. This can help in ensuring that we don't overfit the training data. Since decision tree is relatively fast, I was able to train on entire data batch (10000 samples).

Hyperparameter Selection



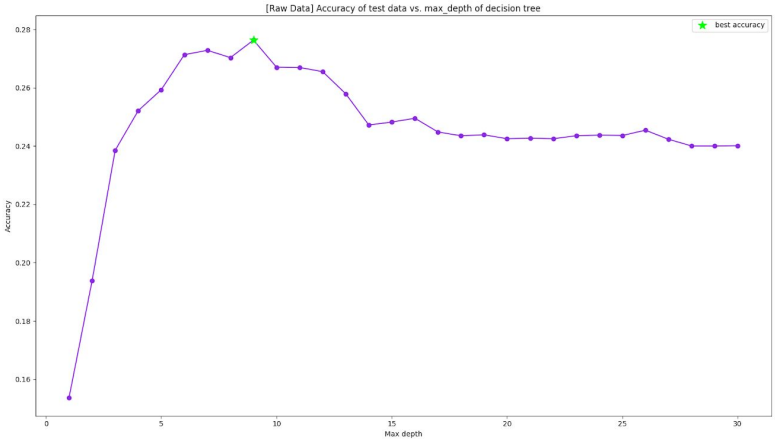


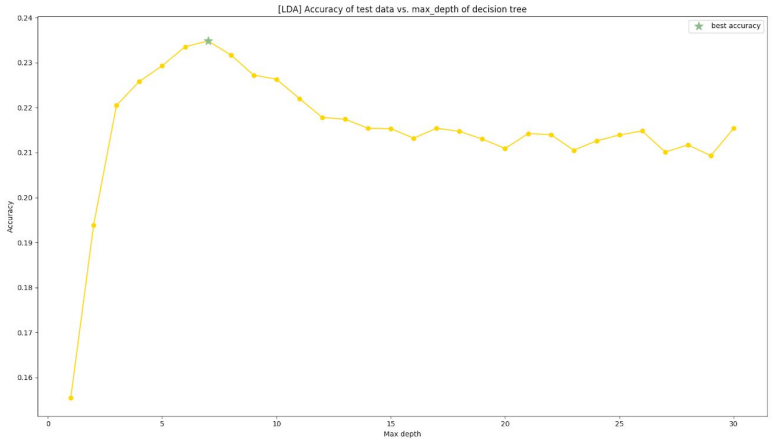
From this we can see that:

- Increasing **max_depth** will monotonically increase the accuracy of the classifier on the training samples, up till 100% accuracy. This is a case of overfitting the training samples, which leads to good performance on the training set but bad performance on any new testing set.
- On validating with the testing set, we can see that increasing **max_depth** upto 8 leads to a better quality decision tree classifier. However increasing **max_depth** any more than that leads to overfitting of the training samples and bad performance on test data.

Results Table

Representation	Best max_depth	Accuracy	F1-score
Raw Data	9	0.2763	0.2763
PCA	8	0.2797	0.2797
LDA	7	0.2348	0.2348


[Raw Data] Accuracy of test data vs. max_depth of decision tree
best accuracy
Accuracy
Max depth
0.28
0.26
0.24
0.22
0.20
0.18
0.16
0
5
10
15
20
25
30


[LDA] Accuracy of test data vs. max_depth of decision tree
best accuracy
Accuracy
Max depth
0.24
0.23
0.22
0.21
0.20
0.19
0.18
0.17
0.16
0
5
10
15
20
25
30