Mounica Mandapati

A20151150

KICKSTARTER PROJECT SUCCESS

CS 5783 Project

**TABLE OF CONTENTS**

Introduction…………………………………………………………..2

Problem statement……………………………………………………2

Data Collection and Description……………………………………..2

Data Cleaning………………………………………………………..4

Feature Engineering………………………………………………….4

Exploratory Data Analysis…………………………………………...6

Modelling and Hyper parameter Tuning……………………………..8

1. K-nearest Neighbor………………………………………….8
2. Random Forest ……………………………………………..10
3. Gradient Boosting…………………………………………..14
4. Deep Neural Network ………………………………………17

Conclusions…………………………………………………………..22

1. **Introduction:**

Crowdfunding has become one of the main sources of initial capital for small businesses and start-up companies that are looking to launch their first products. Websites like [Kickstarter](https://www.kickstarter.com/) and [Indiegogo](https://www.indiegogo.com/) provide a platform for millions of creators to present their innovative ideas to the public. This is a win-win situation where creators could accumulate initial fund while the public get access to cutting-edge prototypical products that are not available in the market yet.

Kickstarter is a community of more than 10 million people comprising of creative, tech enthusiasts who help in bringing creative project to life. Till now, more than $3 billion dollars have been contributed by the members in fueling creative projects. The projects can be literally anything – a device, a game, an app, a film etc.

Kickstarter works on all or nothing basis i.e. if a project doesn’t meet its goal, the project owner gets nothing. For example: if a project’s goal is $500. Even if it gets funded $499, the project won’t be a success.

1. **Problem statement:**

At any given point, Kickstarter has 6,000 live campaigns. It has become increasingly difficult for projects to stand out of the crowd. Of course, advertisements via various channels is by far the most important factor to a successful campaign. However, for creators with a smaller budget, this leaves them wonder,

*"How do we increase the probability of success of our campaign starting from the very moment we create our project on the Kickstarter website?"*

So, the aim of project is to identify factors that help predict the success of kickstarter projects.

1. **Data Collection and Description:**

For the purpose of this project, data is collected from webrobots.io website.

This website has a scraper robot that crawls all Kickstarter projects and collects data in CSV format. The dataset contains 28 columns: 17 numeric, 21 categorical columns.

The description of the useful variables are:

1. Categorical Variables:

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Levels** | **Explanation** |
| id | 179,782 | Unique Identifier for Project |
| name | Text field | Name of the project |
| blurb | Text field | Project short description |
| category | Text field | Category of project |
| State (Target) | 4 | Identifies state of project |
| country | 22 | Origin country of project |
| currency | 14 | Original currency of the projects |
| staff\_pick | 2 | Projects favored by Kickstarter Staff |
| spotlight | 2 | Check the project for Kickstarter spotlight |
| disable\_communication | 2 | Check the project for owner communication option |

1. Continuous Variables:

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Range** | **Explanation** |
| deadline | 05/03/2009- 03/03/2018 | Deadline of the project |
| launched | 04/24/2009- 01/02/2018 | Launch date of the project |
| created\_date | 04/21/2009- 01/02/2018 | Project creation date |
| state\_changed\_date | 05/03/2009- 03/03/2018 | Date of state change for the project |
| backers\_count | 1 - 105,857 | Total count of backers |
| static\_usd\_rate | 0.00877 - 1.7164 | USD conversion rate at the time of project inception |
| pledged\_amount | 0 - 10,266,845.74 | Total amount pledged towards project |
| goal\_amount | 0.01 - 100,000,000 | Goal amount set by project starters |

1. **Data Cleaning:**

Of the available projects, there were some anomalies in the data. So, data cleaning was done as follows:

1. There were around 20,000 duplicate project ids out of around 205,227 project ids. On observing that these duplicate project ids have the same column values, these project ids were removed from further analysis.
2. The dates like **created\_at, deadline** etc. are in interval format. So, they are changed to date format using ‘to\_datetime’ function.
3. The columns **friends, is\_backing, is\_starred, permissions** are having majority missing values. So, they are dropped from further analysis.
4. The column **is\_starrable** is dropped since it has unary distribution. The columns photo, profile, source\_url are also dropped since they don’t have any useful information.
5. There are 4 levels in target, however, only 2 levels (i.e. successful, failed) in target are considered for further analysis. The records with other two levels (i.e. canceled, live) have been omitted.
6. There are 6 projects which do not contain project name. So, these rows are removed from analysis.
7. **Feature Engineering:**

With the available variables, below variables are created. These variables are also included in further analysis to improve model performances.

1. Categorical Variables:

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Levels** | **Explanation** |
| Creator\_name | Text field | Project creator |
| Main\_category | 15 | Category of the project |
| country\_us | 2 | Checks if the country if USA |

1. Continuous Variables:

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Range** | **Explanation** |
| creator\_projects\_count | 1 - 68 | Number of projects created by the project creator |
| days\_btw\_create\_launch | 0 - 3190 | Number of days between project creation and launch date |
| days\_btw\_launch\_stchange | 1 - 97 | Number of days between project launch and state  changed date |
| campaign\_length | 1 - 97 | Number of days between project launch date and deadline |
| blurb\_length | 1 - 151 | Length of project blurb |
| name\_length | 1 - 85 | Length of project name |
| blurb\_words | 1 - 43 | Number of words in project blurb |
| name\_words | 1 - 27 | Number of words in the project name |

Apart from all these variables, there were several text topics created from the project\_name and project\_blurb text field. The following rules were followed while creating text topics:

1. Minimum occurrence of word is 3
2. Removed English stop words
3. Converted all words to lowercase
4. Removed any characters other than alphabets and numbers
5. Removed any word with length less than 3
6. Maximum number of unique words is 300

On the project\_name field, built 5 meaningful text topics and on project\_blurb filed, built 20 meaningful text topics. Also, created another column which shows the dominant text topic for project\_name and project\_blurb for each project.

Example Text topic fields:

|  |  |
| --- | --- |
| **Variable Names** | **Explanation** |
| Blurb\_topic1 | Probability of first Text topic from the project blurb field |
| Dominant\_blurb\_topic | Dominant Text topic from the project blurb field |
| name\_topic1 | Text topic from the project name field |
| Dominant\_name\_topic | Dominant Text topic from the project name field |

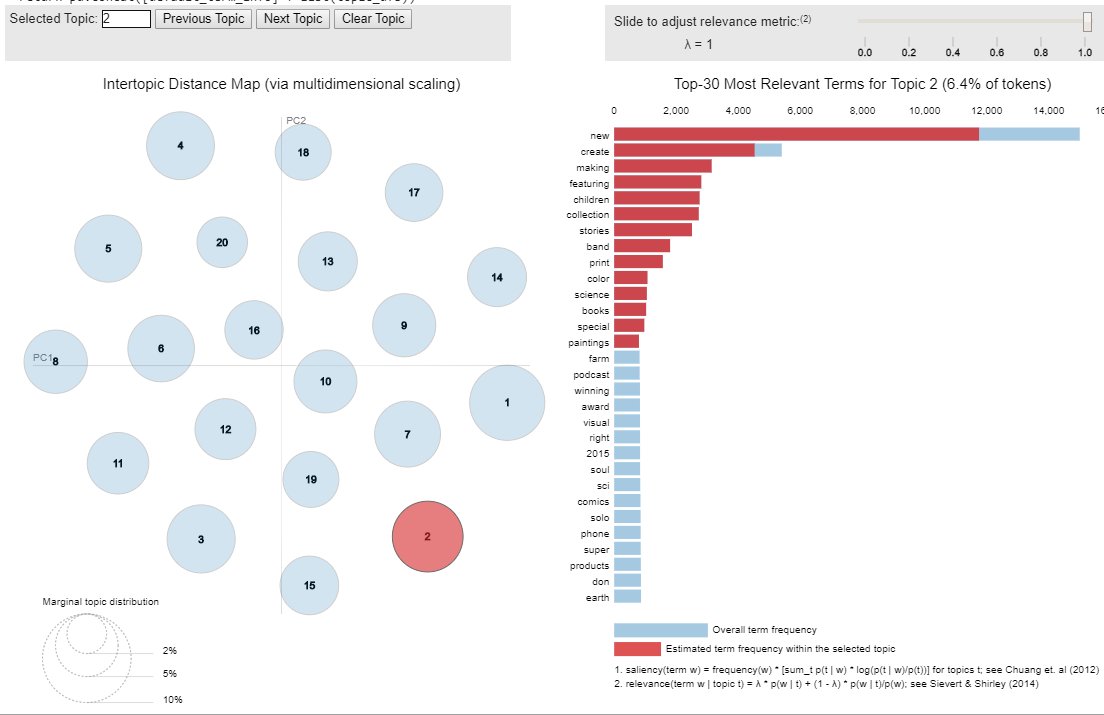
1. **Exploratory Data Analysis:**

Below are the descriptive statistics of the variables available in the dataset.

1. The intertopic distribution map:

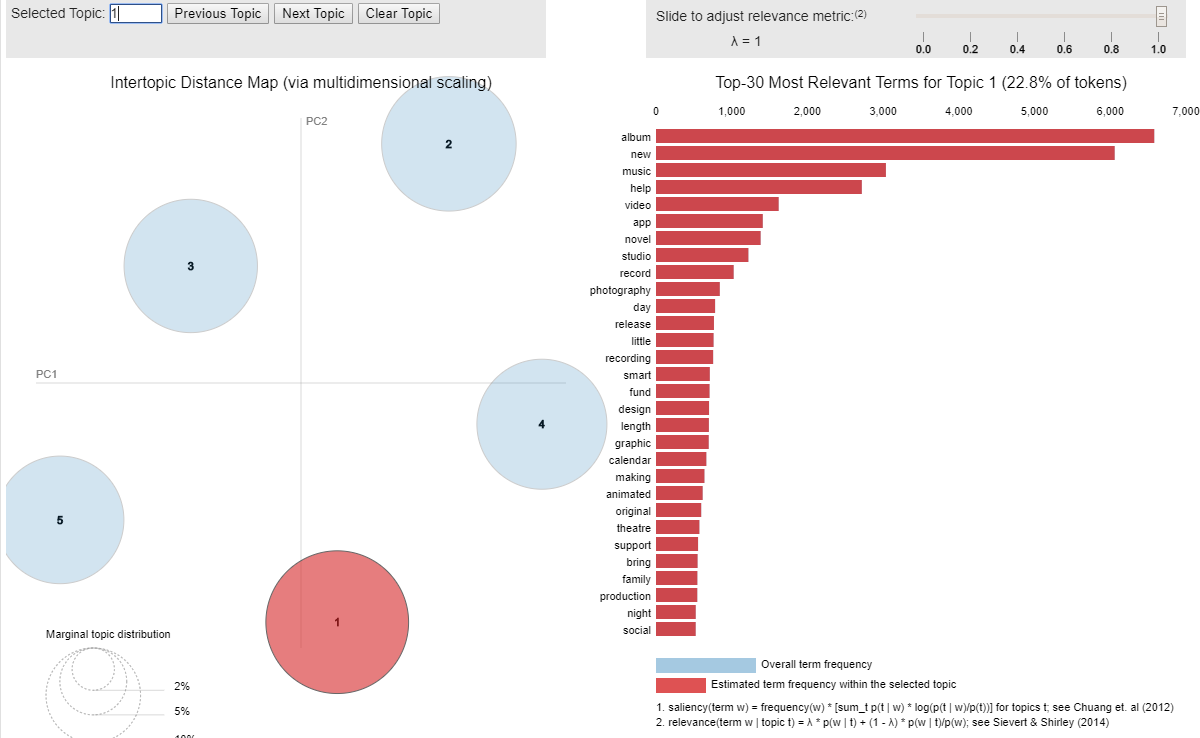
It presents an overall view of the topic model. Different topics are plotted as circles, where overall prevalence was calculated as the areas of the circles. The centers of each topic were determined by computing the distance between topics; multidimensional scaling was used to represent the intertopic distances on a two-dimensional plane. PC1 indicates the transverse axis and PC2 indicates the longitudinal axis.

The bar chart in a descending order of the top 20 most useful terms, for interpreting a topic. The overlaid bars represent a given term’s corpus-wide frequency and the topic-specific frequency. In this bar chart, it is representing the top 20 useful terms of project\_blurb topic 2



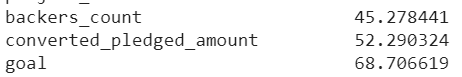
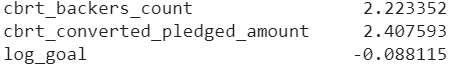
**Figure 1: Intertopic distance Map of project\_blurb**

Similarly, for the project\_name the intertopic distance map and top 20 useful terms for topic 2 are:



**Figure 2: Intertopic distance Map of project\_name**

1. There are interval variables with higher skewness. So, transformed them accordingly using cube root and log transforms to lower their skewness.

**Figure 3: skewness before transforming Figure 4: skewness after transforming**

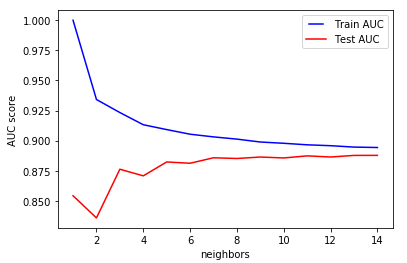
1. Checked the correlation among features and removed 6 features with multicollinearity.
2. On the dataset to reduce dimensionality, used L1 norm to penalize model and make coefficients to zero with lower C value. From the svm.LinearSVC classifier selected 16 variables.
3. **Modelling and Hyper parameter Tuning:**

The dataset after feature engineering and feature selection has 16 variables. The data is also divided into 70% training and 30% validation data, to check the model performance on unseen data. Also, since the data is unbalanced, training set is balanced with under sampling technique to equal proportions of target. Using all the 16 variables, built 4 machine learning techniques on training dataset and tested the performance on validation dataset to predict the success of the Kickstarter projects.

1. **K-nearest neighbor (KNN):** KNN is a distance based algorithm. It assumes that similar data points are near to each other. This algorithm is implemented using KNeighborsClassifier function from sklearn library.

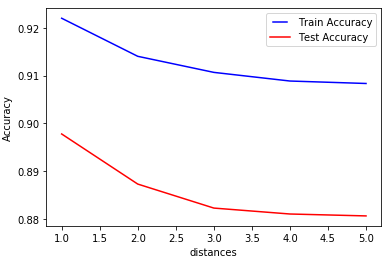
By building KNN with default parameters, the model AUC and misclassification rate are 0.88 and 0.11. The accuracy on test dataset is 88%. Keeping this as baseline model and tuned hyper parameters like n\_neighbors (number of neighbors), p (power parameter for the Minkowski metric)

**Tuning n\_neighbor:** On tuning the number of neighbors from 1 to 15, at 1 it has 100% accuracy, means each sample is using itself as reference, that’s an overfitting case. As the number of nearest neighbors increases, the training accuracy decreases, but it test accuracy is increasing. So, it is more generalized if we use a higher n\_neighbor.



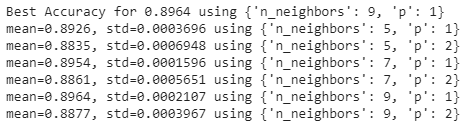
**Figure 5: Accuracy graph for n\_neighbors**

**Tuning p:** On tuning the power parameter for the Minkowski metric from p=1 to 5, it is observed that p=1 is performing better on this data. When p=1, this is equivalent to using manhattan\_distance (l1), and euliddean\_distance (l2) for p=2. For arbitrary p, minkowski distance (l\_p) is used.



**Figure 6: Accuracy graph for p**

**Grid Search:** To exhaustively consider all parameter combinations of n\_neighbors (i.e. k= 5, 7, 9) and power parameter (i.e. p=1, 2), used grid search strategy.



**Figure 7: Grid Search for KNN parameters results**

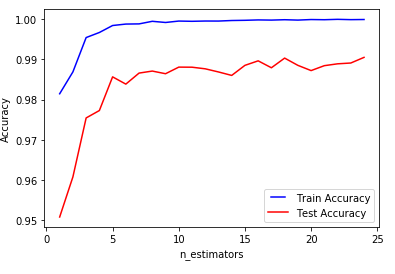
The best model from the grid search is built with 9 nearest neighbors and Manhattan distance. The accuracy of this model on validation data is 90.1%.

1. **Random Forest Classifier**

By building Random Forest with default parameters, the model AUC and misclassification rate on validation data are 0.98 and 0.011. This model accuracy on validation data is 98%. Keeping this as baseline model and tuned hyper parameters like n\_estimators (number of estimators), max\_depth (maximum depth in a forest), min\_samples\_splits (minimum number of splits), min\_samples\_leafs (minimum number of leafs), max\_features (maximum number of features)

**Tuning n\_estimator:** n\_estimators represents the number of trees in the forest. Usually the higher the number of trees the better to learn the data. However, adding a lot of trees can slow down the training process considerably, therefore we do a parameter search to find the sweet spot.

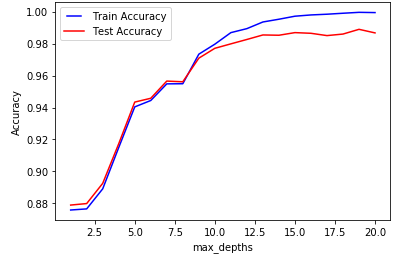
On tuning the n\_estimators from 1 to 25, the test accuracy is highest at 23, 17 estimators. After that it is almost same.



**Figure 8: Accuracy graph for n\_estimators**

**Tuning max\_depth:** max\_depth represents the depth of each tree in the forest. The deeper the tree, the more splits it has and it captures more information about the data. We fit each decision tree with depths ranging from 1 to 20 and plot the training and test errors.

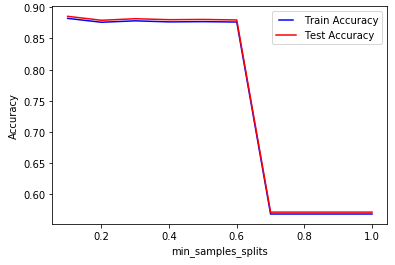
We see that our model overfits for large depth values. The trees perfectly predicts all of the train data, however, it fails to generalize the findings for new data.



**Figure 9: Accuracy graph for max\_depth**

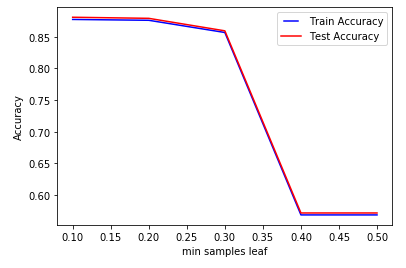
**Tuning min\_samples\_split:** min\_samples\_split represents the minimum number of samples required to split an internal node. This can vary between considering at least one sample at each node to considering all of the samples at each node. When we increase this parameter, each tree in the forest becomes more constrained as it has to consider more samples at each node. Here we will vary the parameter from 10% to 100% of the samples.

We can clearly see that when we require all of the samples at each node, the model cannot learn enough about the data. This is an underfitting case.



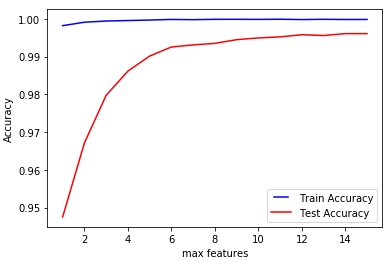
**Figure 10: Accuracy graph for min\_samples\_split**

**Tuning min\_samples\_leaf:** min\_samples\_leaf is the minimum number of samples required to be at a leaf node. This parameter is similar to min\_samples\_splits, however, this describe the minimum number of samples of samples at leafs, the base of the tree. Increasing this value can cause underfitting.



**Figure 11: Accuracy graph for min\_samples\_leaf**

**Tuning max\_features:** max\_features represents the number of features to consider when looking for the best split. This is also an overfitting case. It’s unexpected to get overfitting for all values of max\_features. However, according to sklearn documentation for random forest, the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max\_features features.



**Figure 12: Accuracy graph for max\_features**

**Randomized Search:** Random search is a technique where random combinations of the hyper parameters are used to find the best solution for the built model. It tries random combinations of a range of values. To optimize with random search, the function is evaluated at some number of random configurations in the parameter space.

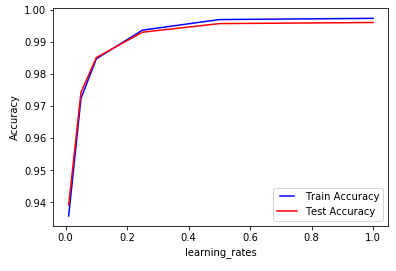
Since there are more parameters in random forest used Randomized search. The parameters used in this are n\_estimators (i.e. 23, 14), max\_features (i.e. ‘auto’,’sqrt’), max\_depth (i.e. 18, 15, none), min\_samples\_Split (i.e. 10, 15), min\_samples\_leaf (i.e. 1, 2). The best model uses none max\_depth, sqrt max\_features, 1 min\_samples\_leaf, 10 min\_samples\_split, 14 n\_estimators. The accuracy of this best model is 98.9 % on validation data.

1. **Gradient Boosting:**

Gradient Boosting trains many models in a gradual, additive and sequential manner. This algorithm identifies the shortcomings of weak learners (eg. decision trees) by using gradients in the loss function.

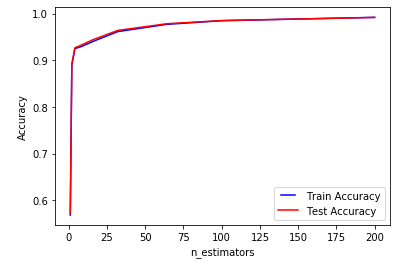
This algorithm is implemented using GradientBoostingClassifier with default parameters. The AUC of this model is 0.98 and misclassification rate is 0.014. Keeping this as baseline model and tuned hyper parameters like learning rate, n\_estimators, max\_depth, min\_samples\_split, min\_samples\_leaf, max\_features.

**Tuning learning rate:** learning rate shrinks the contribution of each tree by learning\_rate. We see that using a high learning rate results in overfitting. For this data, a learning rate of 0.3 is optimal.



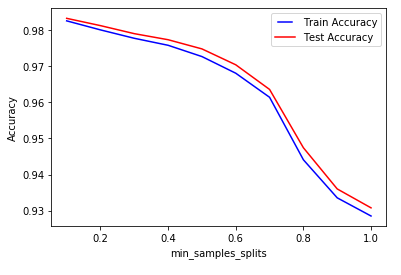
**Figure 13: Accuracy graph for learning rate**

**Tuning n\_estimators:** Increasing the number of estimators may result in overfitting also. In our case, using 50 trees is optimal.



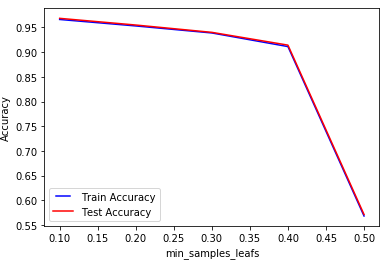
**Figure 14: Accuracy graph for n\_estimators**

**Tuning min\_samples\_split:** The model cannot learn enough about the data when all the samples are used. This is an underfitting case. So, we shouldnot use all the samples at each splits.



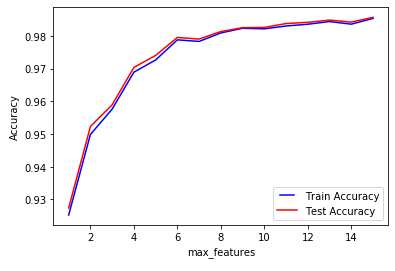
**Figure 15: Accuracy graph for min\_samples\_split**

**Tuning min\_samples\_leaf:** Increasing this value can cause underfitting when the number of samples used at leafs is more. So, less leafs should be used at leafs.



**Figure 16: Accuracy graph for min\_samples\_leaf**

**Tuning max\_features:** Increasing max\_features to consider all of the features results in an overfitting in this case. Using max\_features = 6 seems to get us the optimal performance.



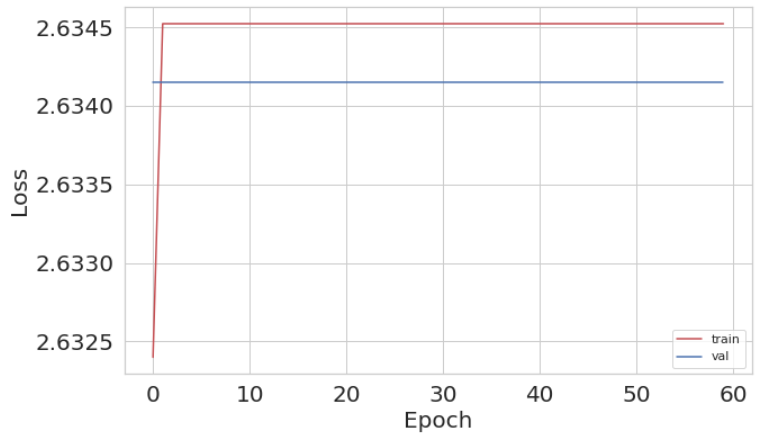
**Figure 17: Accuracy graph for max\_features**

**Randomized Search:** Since there are more parameters in Gradient Boosting, Randomized search is used. The parameters used in this are n\_estimators (i.e. 25, 50), max\_features (i.e. ‘auto’,’sqrt’), max\_depth (i.e. 18, 15, none), min\_samples\_Split (i.e. 10, 15), min\_samples\_leaf (i.e. 1, 2), learning\_rate (i.e. 0.2,0.3). The best model uses none max\_depth, sqrt max\_features, 1 min\_samples\_leaf, 10 min\_samples\_split, 14 n\_estimators. The accuracy of this best model is 98.9 % on validation data.

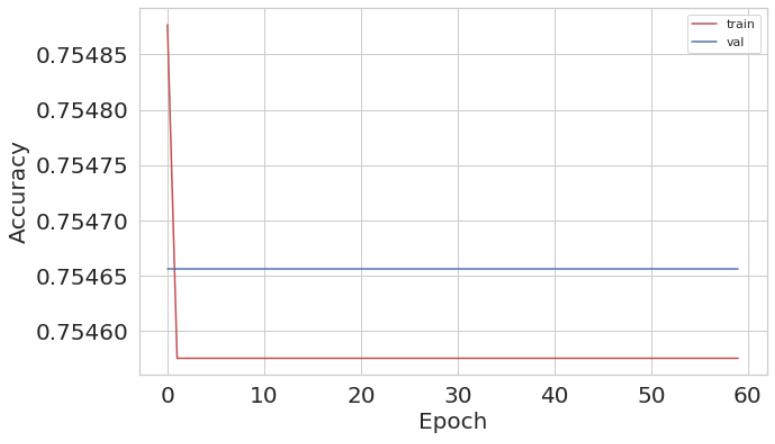
1. **Deep Neural Net (DNN):**

One of the most common optimization algorithms used in DNN is Stochastic Gradient Descent (SGD). The hyper parameters that can be optimized in SGD are learning rate, momentum, decay and nesterov. Learning rate controls the weight at the end of each batch, and momentum controls how much to let the previous update influence the current weight update. Decay indicates the learning rate decay over each update, and nesterov takes the value “True” or “False” depending on if we want to apply Nesterov momentum.

Typical values for those hyper parameters are learning rate=0.01, decay=1e-6, momentum=0.9, and nesterov=True. With these parameters built a 2 hidden layer neuron. The baseline accuracy of this model is 98.2% on test data.

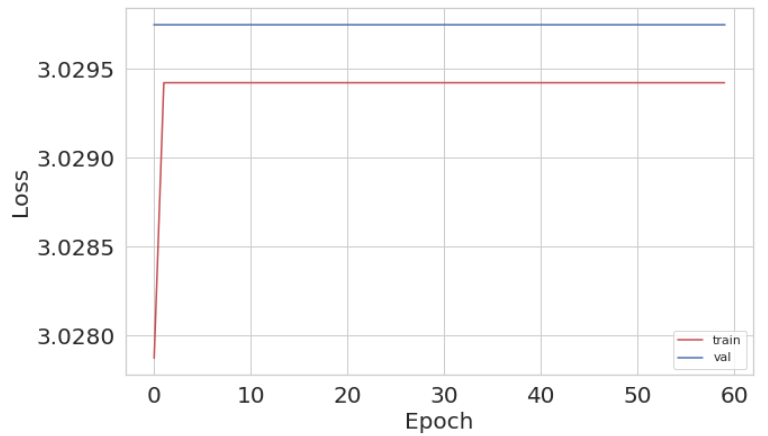


**Figure 18: loss function graph for base model**

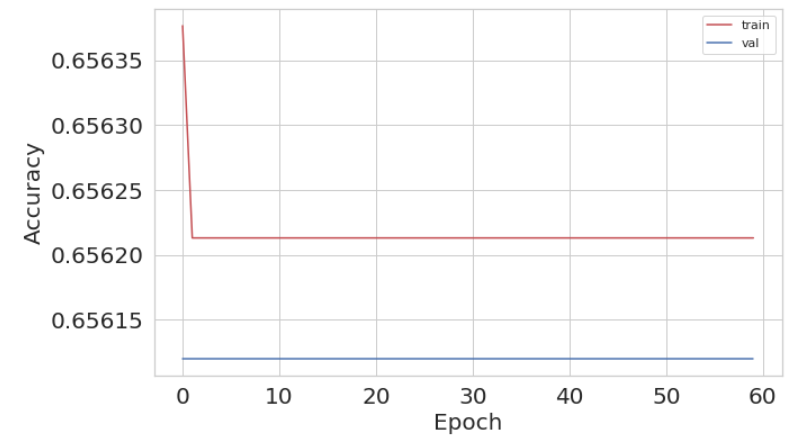


**Figure 19: Accuracy graph for base model**

**Applying custom learning rate:** exp\_decay function is added to the above neuron along with few callbacks. After applying this custom learning rate, the validation loss has increased and accuracy decreased. Even though, the learning rate is smooth, this method did not prove to improve the performance for this data.

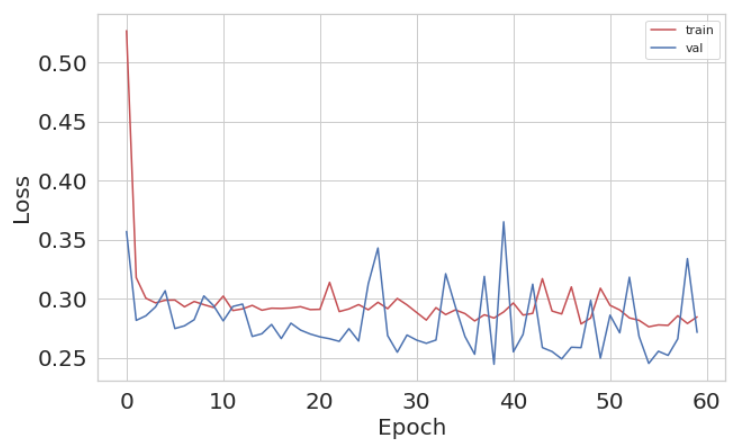


**Figure 20: loss function graph for model with customized learning rate**

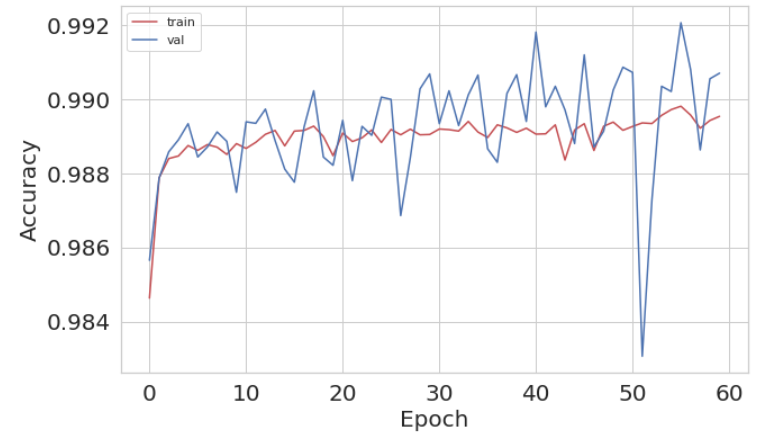


**Figure 21: Accuracy graph for model with customized learning rate**

**Choosing an Optimizer:** Added an RMSprop Optimizer to above model, the results show that there is fluctuations in the results. But loss has drastically decreased and accuracy has improved a lot. So, RMSProp optimizer is performing better than SGD optimizer.



**Figure 22: loss function graph for model with customized optimizer**



**Figure 23: Accuracy graph for model with customized optimizer**

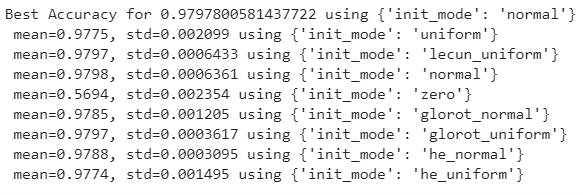
**Varying batch size:** On varying the batch size, it is observed that as batch size increases, the accuracy also increased (but very little).

|  |  |
| --- | --- |
| **Batch size** | **Test accuracy** |
| 200 | 0.9768 |
| 300 | 0.9787 |
| 400 | 0.9792 |

**Varying epochs:** On varying epochs, it is observed that, accuracy is increasing (very little) with increase in epoch.

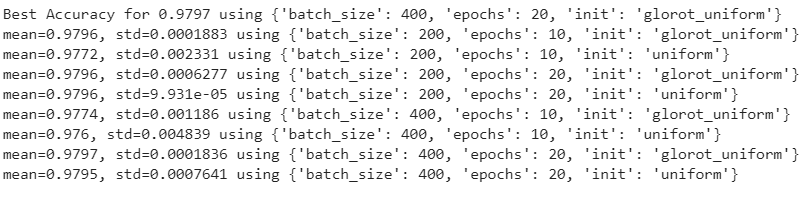
|  |  |
| --- | --- |
| **Epochs** | **Test accuracy** |
| 10 | 0.9787 |
| 20 | 0.9796 |
| 30 | 0.9796 |
| 40 | 0.9798 |

**Varying weight initializations:** Keras has different weight initializations like, lecun\_uniform, uniform, normal, zero, glorot\_uniform, he\_normal etc. All these initializations are tried on this data. For this data, the ‘uniform’, ‘glorot\_uniform’, ’lecun\_uniform’ are performing better with 0.9797 accuracy.



**Figure 24: Grid Search for DNN weight initializations**

**Grid search Optimization:** On using the above best performing parameters with Grid Search, the best accuracy is for batch\_size = 400, epochs = 20, initialization = ‘glorot\_uniform’. This model has an accuracy of 98.9% on validation data.



**Figure 25: Grid Search for DNN with all parameters**

1. **Conclusion:**

For understanding the hyperparamter tuning effect, K-nearest neighbor, Random Forest, Gradient Boosting models were built. Each model has a different set of parameters and need different ways to tune as per above results.

The performance of these 4 models on validation data is as shown in the table. Of all the models KNN has the lowest accuracy on training and test data. The models Random forest, gradient boosting and DNN perform similarly on the data. But, DNN requires more time and computer resources. So, depending on the availability of resources, time these models can be used to get about 99% accuracy.

|  |  |  |
| --- | --- | --- |
| **Model** | **Baseline Accuracy** | **Tuned Model accuracy** |
| K-nearest neighbor | 88.1% | 90.2% |
| Random Forest | 98.1% | 98.9% |
| Gradient Boosting | 98.1% | 98.9% |
| DNN | 98.2% | 98.9% |