Machine Learning Engineer Nanodegree
Capstone Project

Firdos Rehman

July 7th 2018

I. Definition
Project Overview

Based on the characteristics of the leaves provided, machine learning model is built which provides the highest accuracy to find the species of the plants. Out of approximately half a million species of the plants in the world, it is very problematic to classify the different species and it had been seen historically that many of them results in duplicate identifications. We can apply automating the plant recognition so that the it helps it helps in many applications like in preserving and species population tracking, Plant - based medicinal research which many people are choosing as it is safer and lesser side effects, ecological reasons, crop and food supply management

Kaggle hosted this competition for the data science community to use for fun and education. This dataset originates from leaf images collected by
James Cope, Thibaut Beghin, Paolo Remagnino, & Sarah Barman of the Royal Botanic Gardens, Kew, UK. Charles Mallah, James Cope, James Orwell. Plant Leaf Classification Using Probabilistic Integration of Shape, Texture and Margin Features. Signal Processing, Pattern Recognition and Applications, in press. 2013.

Originally the dataset was hosted by UCI machine learning repository.

Problem Statement

This project is to build a model which provides highest accuracy to find the species of plants from characteristics of the leaves.

The objective is to use binary features, including shape, margin & texture, to accurately identify 99 species of plants. Leaves, due to their volume, prevalence, and unique characteristics, are an effective means of differentiating plant species. Finally, examine the errors you're making and see what you can do to improve.

Based on problem I am planning to use the supervised classification machine learning algorithms logistic regression, random forest and the neural network model. I will compare the results we get from these models and choose the best classifier.

Metrics

In Classification model we typically measure the performance of the model by using accuracy which is defined as the number of correct predictions from all the predictions made. In addition to finding the accuracy, submissions are evaluated using the multi-class logarithmic loss. The formula is then,

$$\text{logloss} = -1/N \sum_{i=1}^{N} \sum_{j=1}^{M} yij \log(Pij)$$

where N is the number of images in the test set, M is the number of species labels, log is the natural logarithm, yij is 1 if observation i is in class j and 0 otherwise, and p ij is the predicted probability that observation i belongs to class j.

The submitted probabilities for a given device are not required to sum to one because they are rescaled prior to being scored (each row is divided by the row sum), but they need to be in the range of [0, 1]. In order to avoid the extremes of the log function, predicted probabilities are replaced with

$$max(min(p, 1 - 10^{-15}), 10^{-15})$$

II. Analysis

Data Exploration

The dataset is taken from Kaggle(URL is provided below)For each feature, a 64-attribute vector is given per leaf sample. Based on the characteristics of the leaves provided, machine learning model is built which provides the highest accuracy to find the species of the plants. The dataset is available in Kaggle.

https://www.kaggle.com/c/leaf-classification/data

Note that of the original 100 species, we have eliminated one on account of incomplete associated data in the original dataset.

File descriptions¶

- train.csv - the training set
- test.csv - the test set

Data fields¶

- id - an anonymous id unique to an image
- margin_1, margin_2, margin_3, ..., margin_64 - each of the 64 attribute vectors for the margin feature
- shape_1, shape_2, shape_3, ..., shape_64 - each of the 64 attribute vectors for the shape feature
- texture_1, texture_2, texture_3, ..., texture_64 - each of the 64 attribute vectors for the texture feature

train.csv: Below is the dataset displaying the Id, species and the features

	id	species	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	...	texture55	texture56	texture57	texture58	textur
0	1	Acer_Opalus	0.007812	0.023438	0.023438	0.003906	0.011719	0.009766	0.027344	0.0	...	0.007812	0.000000	0.002930	0.002930	0.035
1	2	Pterocarya_Stenoptera	0.005859	0.000000	0.031250	0.015625	0.025391	0.001953	0.019531	0.0	...	0.000977	0.000000	0.000000	0.000977	0.023
2	3	Quercus_Hartwissiana	0.005859	0.009766	0.019531	0.007812	0.003906	0.005859	0.068359	0.0	...	0.154300	0.000000	0.005859	0.000977	0.007
3	5	Tilia_Tomentosa	0.000000	0.003906	0.023438	0.005859	0.021484	0.019531	0.023438	0.0	...	0.000000	0.000977	0.000000	0.000000	0.020
4	6	Quercus_Variabilis	0.005859	0.003906	0.048828	0.009766	0.013672	0.015625	0.005859	0.0	...	0.096680	0.000000	0.021484	0.000000	0.000

5 rows × 194 columns

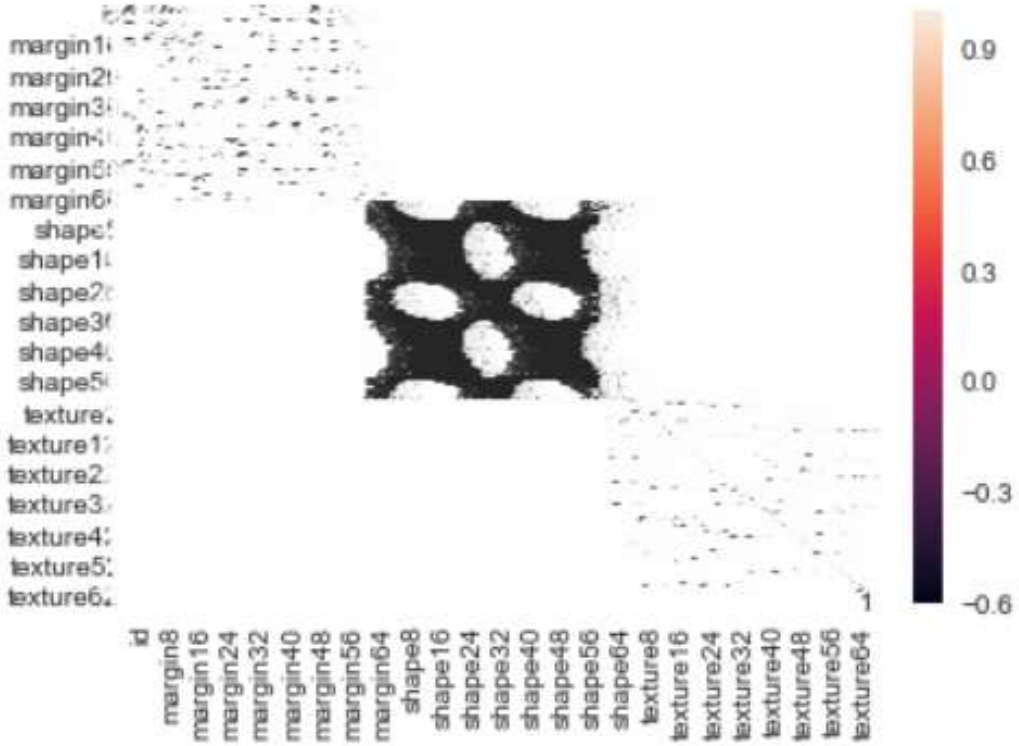
Statistical summary of the dataset:

	id	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	...	texture55	texture56
count	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	...	990.000000	990.000000
mean	799.595960	0.017412	0.028539	0.031988	0.023280	0.014264	0.038579	0.019202	0.001083	0.007167	...	0.036501	0.005024
std	452.477568	0.019739	0.038855	0.025847	0.028411	0.018390	0.052030	0.017511	0.002743	0.008933	...	0.063403	0.019321
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000
25%	415.250000	0.001953	0.001953	0.013672	0.005859	0.001953	0.000000	0.005859	0.000000	0.001953	...	0.000000	0.000000
50%	802.500000	0.009766	0.011719	0.025391	0.013672	0.007812	0.015625	0.015625	0.000000	0.005859	...	0.004883	0.000000
75%	1195.500000	0.025391	0.041016	0.044922	0.029297	0.017578	0.056153	0.029297	0.000000	0.007812	...	0.043701	0.000000
max	1584.000000	0.087891	0.205080	0.156250	0.169920	0.111330	0.310550	0.091797	0.031250	0.076172	...	0.429690	0.202150

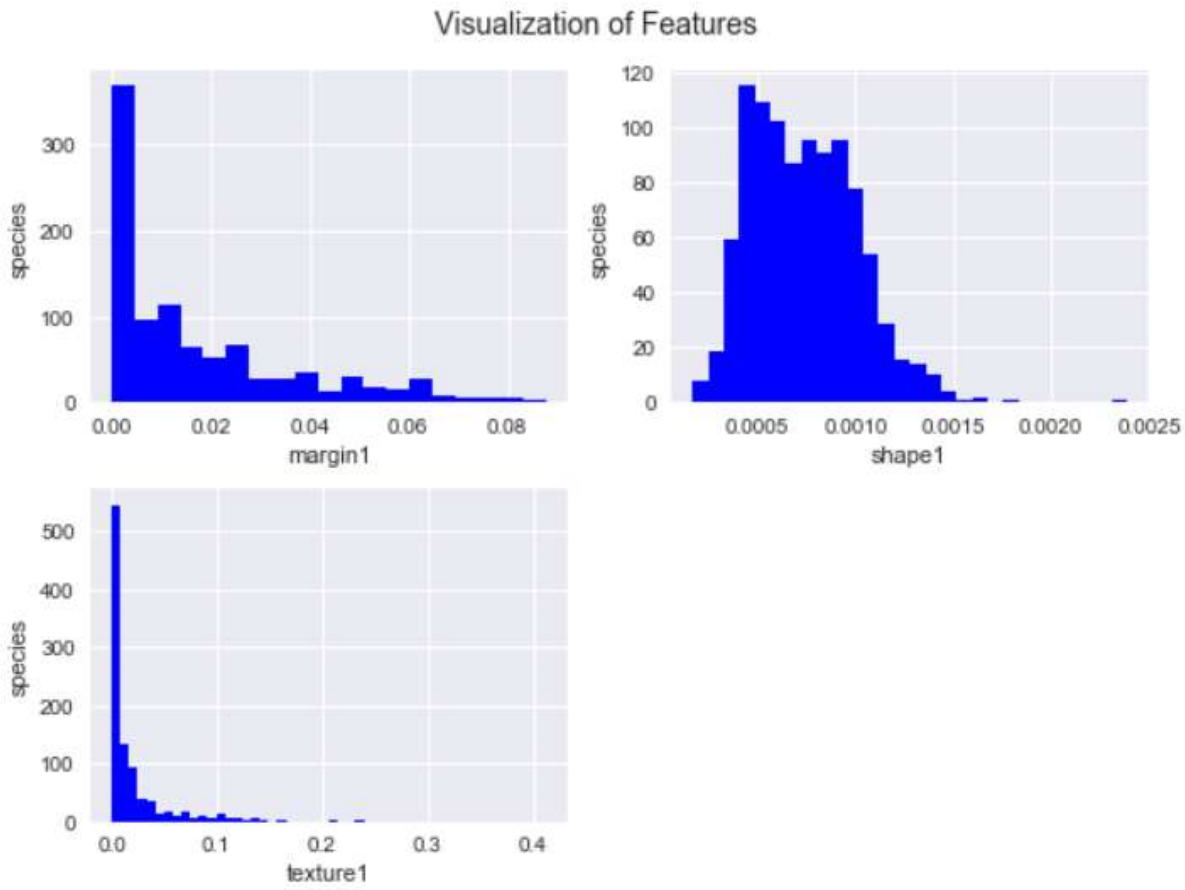
Exploratory Visualization

In the visualization provided below is the correlation matric of the features using the heatmap

Correlation matrix



Below is the visual plot of data considered which explains the Horizontal axis as features (margin1, shape1 and texture1) and vertical axis as species.



Algorithms and Techniques

Equal Probability Model which shows that there is an equal probability of being from each plant or any species. This is baseline benchmark model.

Then I will use Logistic Regression and Random Forest models. I will use 2 classifiers and then see which gives me the best accuracy and the best fit.

Logistic Regression: Logistic Regression Logistic regression measure the relationships between categorical dependent variables and one or more independent variables. Logistical regression models use probability scores as the predicted values of the dependent variables. As it has low variance, it will not be overfitting. Performs well with few categorical variables, Good Probabilistic interpretation, doesn't have to worry about features being correlated. Logistic regression is good for qualitative and quantitative data.

Random Forest: Random forest is one of the mostly used and powerful supervised machine learning algorithm. It is an ensemble model mostly used in classification and regression. It is the forest of many decision trees trained mostly using the bagging method. While splitting the node it searches for the random subset of features instead of searching for the most important features. The random set of features are considered by the random forest algorithm at each node which results in diversity and giving a better result. You can even

make trees more random, by additionally using random thresholds for each feature rather than searching for the best possible thresholds. Because of the robust nature, it is good in prediction and gives better accuracy. It won't overfit the model. It works very well with large data sets with high dimensionality. It reduces variance and improve performance. The important hyperparameters used to increase predictive power of the model in sklearn are "n_estimators" which is the number of trees, "max_features" the number of features, "min_sample_leaf" the minimum number of leaves. To increase the speed of the model "n_jobs" tells engine the processors it is allowed to use, "random_state" which makes model'soutput replicable with same parameters and training data, "oob_score" out of the bag sampling.

Neural Network model: I am going to build a neural network model(Feed Forward Neural Network) using Keras as my base model. Using the sequential model, I will be adding dense layers and a drop out layer.

Dense layer is a regular layer of neurons which receives input from all the neurons and it is connected to the next layer. The layer has matrix, bias and acivations. I will be using 'Relu' and 'softmax' for activation.

Dropout is a regularization technique used to reduce the model complexity and prevent overfitting. In keras the values for the dropout are between 0 and 1. It drops the fraction of neurons which helps to avoid overfitting.

In feedforward neural network which is the simplest of neural networks, the information moves only in one direction from the input nodes to the hidden layers and the output nodes. Parameters like number of features, dropout, check losses, optimizers, number of epochs will be added or reduced and adjusted to achieve optimum result. The sequential mode is best suited for classification. To compile I will use loss='categorical_crossentropy', optimizer='adam', and metrics as 'accuracy'. Finally use metrics to check the performance.

Benchmark

Equal Probability Model which shows that there is an equal probability of being from each plant or any species. This will be the baseline benchmark model. Then I will use Logistic Regression and Random Forest models. I will use 2 classifiers and then see which gives me the best accuracy and the best fit.

```
==============================
LogisticRegression
****Results****
Accuracy: 65.6566%
Log Loss: 4.167567814778244
==============================
RandomForestClassifier
****Results****
Accuracy: 90.4040%
Log Loss: 1.4464188353593546
==============================
```

Based on the benchmark model the accuracy using the random forest is better (90.4%) compared to the Logistic Regression (65.6%)

III. Methodology

Data Preprocessing

Dropping Columns - The id and species columns were dropped from the training data set and id column is dropped from the testing set.

	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	margin10	...	texture55	texture56	texture57	texture58	texture59
0	0.007812	0.023438	0.023438	0.003906	0.011719	0.009766	0.027344	0.0	0.001953	0.033203	...	0.007812	0.0	0.00293	0.00293	0.035156

The names of the species were in the form of string in the training data was converted integer format using Label encoding

Stratified Shuffle Split: Cross validation iterator is used to split the training dataset. Stratified Shuffle Split cross validation iterator provides train, test indices to split data in training test and testing set. This cross-validation object is a combination of Stratified K Fold and Shuffle Split, which returns stratified randomized folds are made by preserving the percentage of samples for each class.

20% of the training data was reserved as a validation data and split into y-test and x-test. Hence in whole data was split into four parts. This step is necessary as it makes the model more generalized and hence further preventing it from overfitting.

Data preparation using One-hot encoding:

One Hot Encoding: In one-hot encoding only one bit of the state vector is asserted for any given state. All other state bits are zero. Thus if there are *n* states then *n* state flipflops are required. As only one bit remains logic high and rest are logic low, it is called as One-hot encoding.

It is faster than other encoding techniques. Speed is independent of number of states, and depends only on the number of transitions into a particular state. In this model (y_train) which was already labeled encoded has been further hot encoded before training the model. In keras, one hot technique can be done by using function to_categorical.

```
# one-hot encode the labels
y_train_nn = np_utils.to_categorical(y_train, 99)
y_test_nn = np_utils.to_categorical(y_test, 99)
```

```
One-hot labels:
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0.]
```

Implementation

After the completion of the benchmark model, sequential layered for neural network(Feed forward neural network) is developed. Sequential model is a linear stack of layers Input dimensions should be equal to the number of features The feedforward is a simple neural network in which the information moves in one direction from input node to the hidden node and then the output node without using any loops and cycles

Model training and prediction:

- Sequential model is selected using Keras
- Added dense layers with units and classes
- Between the dense layers a dropout layer is added in-order to avoid overfitting
- Activation functions used are 'Relu' and 'softmax'
- Compile using the logloss function to find he error
- Train the model
- Evaluate the accuracy
- The hidden and input layers parameters are decided by using different number to achieve highest accuracy.

```
from keras.models import Sequential
from keras.layers import Dense, Dropout

# define the model
model = Sequential()
model.add(Dense(192, input_shape=X_train.shape[1:], activation='relu'))
model.add(Dropout(0.8))
model.add(Dense(99, activation='softmax'))

# summarize the model
model.summary()
```

```
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam',
              metrics=['accuracy'])


# train the model
hist = model.fit(X_train, y_train_nn, batch_size=128, epochs=200,
          validation_split=0.2, callbacks=[],
          verbose=1, shuffle=True)

# evaluate test accuracy
score = model.evaluate(X_test, y_test_nn, verbose=0)
accuracy = 100*score[1]

# print test accuracy
print('Test accuracy: %.4f%%' % accuracy)
```

Refinement

Added dense layer and dropouts with different parameters like features, dropout, epochs get the highest accuracy avoiding overfitting.

As shown below:

128 features 500 epochs

```
Epoch 500/500
633/633 [==============================] - 0s 56us/step - loss: 0.1395 - acc: 0.9826 - val_loss: 0.2796 - val_acc: 0.9371
Test accuracy: 93.4343%
```

192, 500 epochs

```
Epoch 500/500
633/633 [==============================] - 0s 104us/step - loss: 0.0746 - acc: 0.9937 - val_loss: 0.2538 - val_acc: 0.9560
Test accuracy: 93.4343%
```

```
Epoch 400/400
633/633 [==============================] - 0s 91us/step - loss: 0.0116 - acc: 0.9984 - val_loss: 0.2023 - val_acc: 0.9686
Test accuracy: 94.9495%
```

```
Epoch 600/600
633/633 [==============================] - 0s 88us/step - loss: 0.0024 - acc: 1.0000 - val_loss: 0.2127 - val_acc: 0.9686
Test accuracy: 94.9495%
```

128, 600 epochs

```
Epoch 600/600
633/633 [==============================] - 0s 92us/step - loss: 0.0916 - acc: 0.9905 - val_loss: 0.2606 - val_acc: 0.9497
Test accuracy: 92.4242%
```

64,600

```
Epoch 600/600
633/633 [==============================] - 0s 69us/step - loss: 0.3058 - acc: 0.9352 - val_loss: 0.3774 - val_acc: 0.9057
Test accuracy: 91.9192%
```

192,400, dropout 0.6

```
Epoch 400/400
633/633 [==============================] - 0s 139us/step - loss: 0.2348 - acc: 0.9637 - val_loss: 0.3228 - val_acc: 0.9308
Test accuracy: 92.4242%
```

192,400, dropout 0.7

```
Epoch 400/400
633/633 [==============================] - 0s 186us/step - loss: 0.3890 - acc: 0.9163 - val_loss: 0.3697 - val_acc: 0.9308
Test accuracy: 91.4141%
```

192, 600 epochs, dropout 0.7

```
Epoch 600/600
633/633 [==============================] - 0s 112us/step - loss: 0.0718 - acc: 0.9874 - val_loss: 0.2209 - val_acc: 0.9560
Test accuracy: 93.9394%
```

200epochs

```
Epoch 200/200
633/633 [==============================] - 0s 183us/step - loss: 0.9142 - acc: 0.7915 - val_loss: 0.7972 - val_acc: 0.8868
Test accuracy: 87.8788%
```

```
Epoch 200/200
633/633 [==============================] - 0s 121us/step - loss: 0.3461 - acc: 0.9226 - val_loss: 0.3797 - val_acc: 0.9182
Test accuracy: 91.9192%
```

```
Epoch 200/200
633/633 [==============================] - 0s 110us/step - loss: 0.1891 - acc: 0.9700 - val_loss: 0.2876 - val_acc: 0.9434
Test accuracy: 93.9394%
```

0.8 dropout

```
Epoch 200/200
633/633 [==============================] - 0s 180us/step - loss: 1.2641 - acc: 0.6777 - val_loss: 0.9954 - val_acc: 0.8616
Test accuracy: 86.8687%
```

```
Epoch 200/200
633/633 [==============================] - 0s 413us/step - loss: 0.6135 - acc: 0.8499 - val_loss: 0.4496 - val_acc: 0.9182
Test accuracy: 91.9192%
```

```
Epoch 200/200
633/633 [==============================] - 0s 208us/step - loss: 0.4420 - acc: 0.8799 - val_loss: 0.3207 - val_acc: 0.9308
Test accuracy: 92.4242%
```

```
Epoch 200/200
633/633 [==============================] - 0s 128us/step - loss: 0.2694 - acc: 0.9384 - val_loss: 0.2670 - val_acc: 0.9434
Test accuracy: 92.9293%
```

```
Epoch 200/200
633/633 [==============================] - 0s 227us/step - loss: 0.2043 - acc: 0.9494 - val_loss: 0.2476 - val_acc: 0.9434
Test accuracy: 92.9293%
```

```
Epoch 200/200
633/633 [==============================] - 0s 211us/step - loss: 0.1923 - acc: 0.9510 - val_loss: 0.2313 - val_acc: 0.9497
Test accuracy: 93.4343%
```

IV. Results

Model Evaluation and Validation

The main aim of this project is to achieve the best accuracy and lower the logloss value. During the initial stages the training accuracy and the validation accuracy was higher and causing the overfitting. After making changes to the number of inputs, epochs and dropout values we were able to achieve the Test accuracy of 93.43% which much better than the benchmark model.

We have the accuracy of the training set, the validation set and the test set. Generally speaking the loss of test set should be lower than the loss of the validation set and training set. If I take an example here where the training accuracy is 99% and test accuracy is 85% it is overfitting. Here I will be comparing the test accuracy, the validation accuracy and the training accuracy. Test accuracy is 93.43%, validation accuracy is 94% and the training

accuracy which is 95%. As we have seen the training accuracy, the validation accuracy and the test accuracy are close which is an indication that it is not overfitting so the model is fairly robust.

```
Epoch 200/200
633/633 [==============================] - 0s 211us/step - loss: 0.1923 - acc: 0.9510 - val_loss: 0.2313 - val_acc: 0.9497
Test accuracy: 93.4343%
```

Justification

Based on the results of the bench mark classifiers, the accuracy was 90.4% for the Random Forest and for Logistic Regression it was 65.6%. But the neural network model achieved the accuracy of 93.43% which I was aiming for. I realized that the project can be made better in many ways which I mentioned in the improvement section.

V. Conclusion

Free-Form Visualization

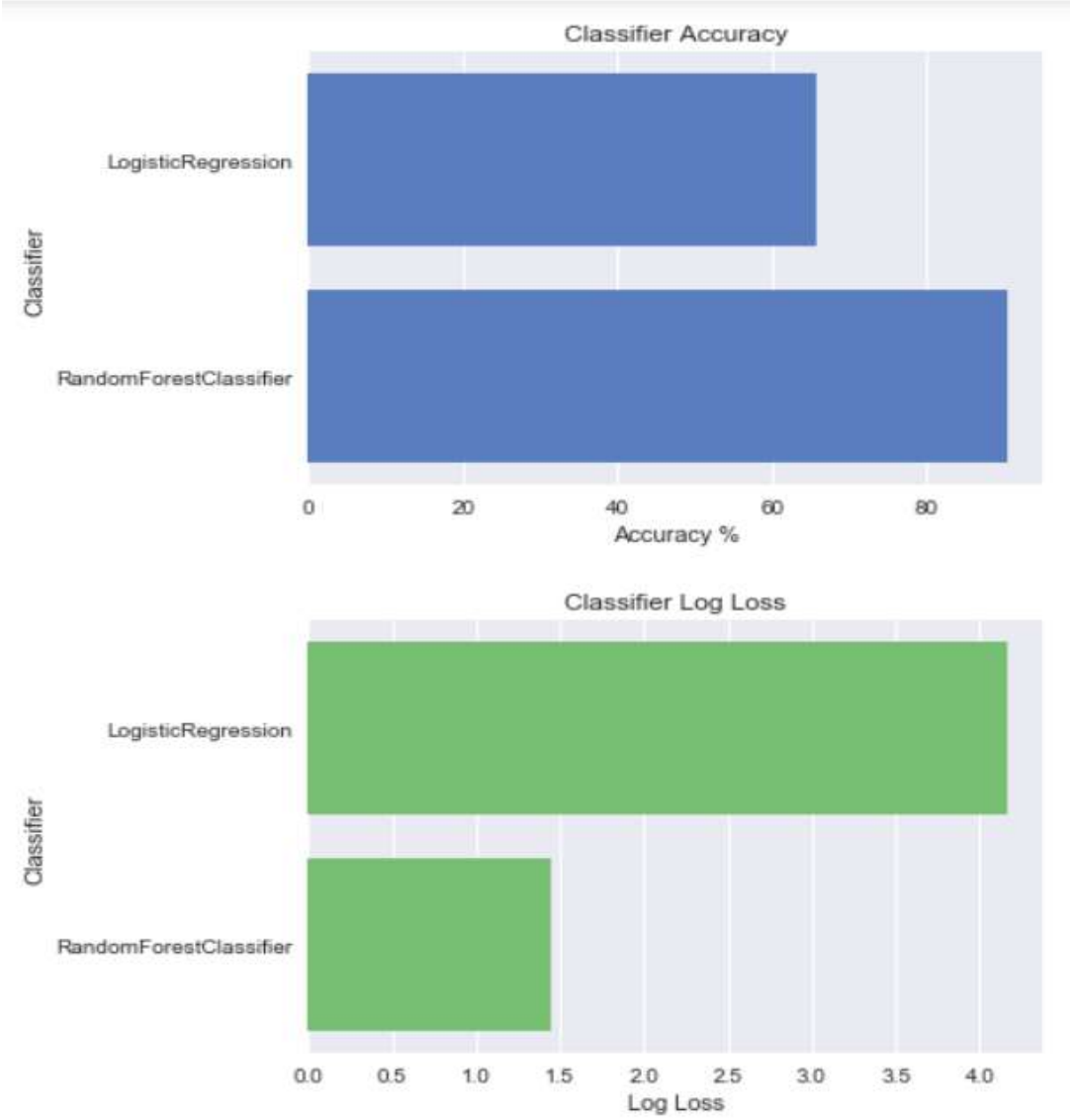


Table comparing the accuracy and logloss

Model	Accuracy	Logloss
Logistic Regression	65.6%	4.16
Random - Forest	90.4%	1.44

Reflection

The process for the end- to -end problem is summarized as follows:

- Identify problem and collect the dataset. The public dataset that I am using from Kaggle
- Preprocess the data
- Based on the benchmark identify the classifiers
- Based on the problem statement and the evaluation metrics, identify the algorithms and techniques to be used
- Check the visualizations of the dataset
- Finish the data processing
- Implement the model and train the model
- Refine the model based on the results and the evaluation metrics

  After completion of the benchmark model, starting the neural network modeling and building the base model was interesting
  To improve the accuracy, changing the different parameters, epochs was a bit difficult and time consuming.
  Yes, the final model and solution fit the expectation for the problem.

Improvement

After scanning through the different kernels in Kaggle I can say that further improvements can be made on this project

I have proposed in my proposal that I will be using the convolutional neural network. But in the interest of time, I was only able to do the base neural network model using the images. As an improvement to this I will do convolutional neural networks in future as an improvement to the base neural network model.

More visualizations and graphs for the to show train error vs number of iterations and graph to show model accuracy can be made.

References:

1. Kaggle Competition: https://www.kaggle.com/c/leaf-classification

2. Kaggle Kernel: https://www.kaggle.com/jeffd23/10-classifier-showdown-in-scikit-learn

3. Keras: https://keras.io/getting-started/sequential-model-guide/

4. Scikit-Learn: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html

5. https://machinelearningmastery.com/custom-metrics-deep-learning-keras-python/