Leaf Classification

Machine Learning Nanodegree Capstone

Daniel Beasley

April 11th, 2017

I. Definition

Project Overview

The goal of this project is to classify leaves from various species of plants. This is a difficult task for humans to perform as there are estimated to be nearly half a million species of plants in the world. A machine learning-based approach to the classification process can have many benefits. It will make the study of plants much easier for researchers. Plant biologists will be able to spend less time identifying plant species and be able to focus more resources on the analysis of data. Automation will allow for quickly tracking the spread of plant species, estimating population counts around the globe, plant-based medicinal research, and better crop and food supply management.[1]

Problem Statement

The problem is to correctly classify 99 different species of plants from their leaves. This is a multi-class classification problem with 99 output classes. There are two datasets with which to apply classifiers. The first dataset is 1584 images of leaves - 16 images for each of the 99 species. The second is a csv file of extracted features. There are three sets of features in the csv file. The shape features describe the shape of the leaf, the margin features describe the fine details of the leaf, and the texture features describe the interior structure of the leaf. Each of the three feature sets contain 64 columns, for a total of 192 extracted features.[2]

In order to complete this problem, the extracted features will be used to build a leaf classifier. The extracted features contain information about the internal structure of the leaves, which is not shown in the image files (shown on the next page).

An adequate solution to this problem is having at least 99% accuracy on a validation set and under 0.1 logarithmic loss on the test set. The validation set will be created during training and the testing set is provided by Kaggle.

Metrics

Accuracy, as a metric, is not very effective in unbalanced datasets. For example, in fraud detection, positive cases are rare, and so a classifier marking all cases as negative would have a very high accuracy, but also be completely useless. However, the dataset for this problem is balanced across the 99 classes, and so accuracy will effectively quantify the results.

Kaggle uses the metric logarithmic loss (log loss) to grade the testing set. Essentially, for each of the test instances, the classifier predicts the probability that the instance is one of the 99 classes. Log loss is then calculated as the negative sum of the average products between true classes and the log of predicted probabilities. This is an effective metric for multi-class problems because, in practice, it serves to penalize incorrect class predictions.[3]

II. Analysis

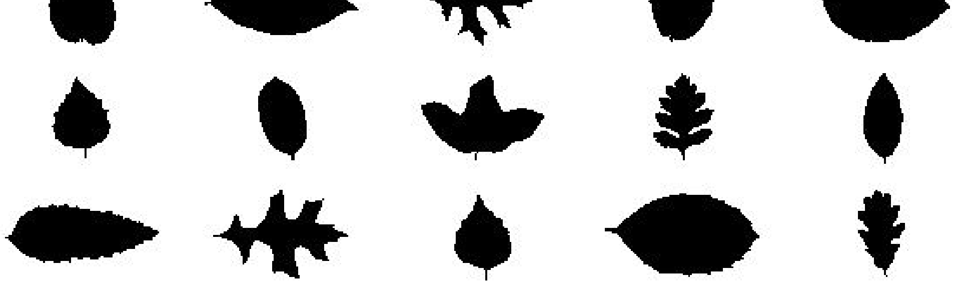
Data Exploration

From the exploratory analysis, we can uncover some basic facts about the data. The training set has 990 samples of 194 features and the test data has 594 samples of 193 features, with the ‘species’ label absent from the test set. For each of the 99 classes, ten samples are in the training set and six are in the test set.

The features are labelled ‘margin1’ through ‘margin64’, ‘shape1’ through ‘shape64’, and ‘texture1’ through ‘texture64’, along with an ID column. The ID column corresponds to the image files from the image data set. The margin values range from 0 to 0.4 with a mean of 0.016, the shape values range from 1e-6 to 0.003 with a mean of 6e-4, and texture values range from 0 to 0.55 with a mean of 0.016. If these values are not normalized, distance-based classifiers like support vector machines and k-nearest neighbours won’t work. Therefore, they are normalized to the range [0, 1].

The species information is the Latin name of the plant. The classifiers can only predict numbers, so this information is given a unique encoding as a number from 0 to 98, and the Latin names are dropped from the data.

Here is a sample of some unique species of plants’ leaves. Clearly, some leaves look very similar, and are indistinguishable to the untrained eye.



Algorithms and Techniques

The algorithms used are decision trees, random forests, AdaBoost, logistic regression, stochastic gradient descent, support vector machines, and a deep neural network.

Decision trees split features by asking binary questions that maximize the information gain. Questions that maximize information gain minimize entropy by attempting to split the data roughly in half (or however entropy is minimized). For example, if one were to make a plot of human adult height, it would likely be bimodal, with peaks at the mean male and mean female height. A point estimate for height would fall between the two peaks and have a large standard deviation. If we were able to ask a question with a decision tree, the first question we would ask is, 'is the person male or female?' This splits the data to give a better estimate with a smaller standard deviation among both groups. The decision tree continues to split the groups into smaller ones until a set of criteria exist to better estimate the parameter in question. Random forests use the same method, but train multiple trees that ask different questions, and average the results.

The AdaBoost classifier is a boosting algorithm. Boosting is a way of combining weak learners, algorithms that perform only slightly better than chance, into a strong learner, one which is well-correlated with the true classification.[4] AdaBoost trains on the original dataset, and fits highly weighted copies of the classifier on the more difficult training examples.[5]

Logistic regression and support vector machines are linear models that work to separate classes by drawing straight lines between them. Stochastic gradient descent implements a loss function with linear models to generate K classifiers, where K is the number of classes. Each of the K classifiers is binary, and discriminates between it and the other K-1 classes.[6]

A deep neural network multiplies feature vectors by several weight matrices, with the last weight matrix having K columns corresponding to the K classes. After each training batch is passed through the network, logits (predictions) are calculated and compared to the true values using a loss function. An error term, representing the difference between the logits and true values, is propagated backwards through the neural network in order to update the weight values in the matrices.

Benchmark

As a sample solution to this problem, Kaggle provides a submission in which each test instance is assigned equal probability of being each of the 99 classes. Any submission scoring better than this can be said to be better than random, and so this is a good benchmark. The sample submission scores 4.6 on logarithmic loss.

III. Methodology

Data Preprocessing

There are no missing values to be imputed in the data set. The only preprocessing to be done was to normalize the data to the range [0, 1], and to convert the categorical species labels to integers.

Implementation

The analysis proceeds as follows:

- Step 1: train a variety of classifiers from scikit-learn on the pre-extracted features. Grid search will be used lightly to make sure some important parameters are in the right neighbourhood.
- Step 2: train a TensorFlow neural network on the features. It will be kept shallow and relatively simple to get a feel for how neural networks perform on this data set.
- Step 3: select the best-performing algorithms and perform a thorough optimization. For a scikit-learn algorithm, this will involve providing several values for parameters in the grid search. For a neural network, it will involve adding more layers and adjusting hyperparameters in order to better the algorithm's performance.

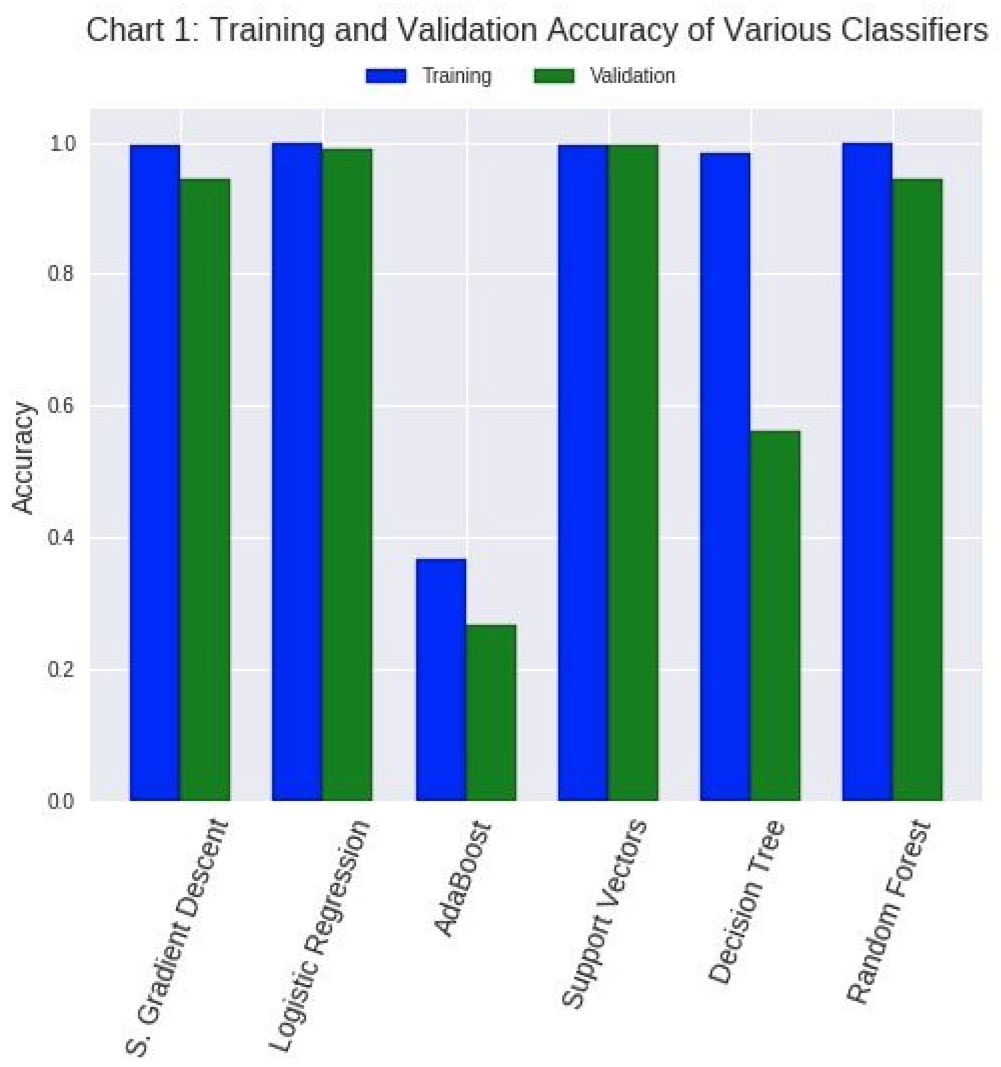
Let's examine each step in more detail.

In Step 1, we train decision trees, random forests, AdaBoost, logistic regression, stochastic gradient descent, and support vector machines on the training data. Rather than just using the base classifier, grid search is used with a few choices for some important parameters. This is done to give a better indication of how each algorithm performs on the data. The grid search parameters can be seen below.

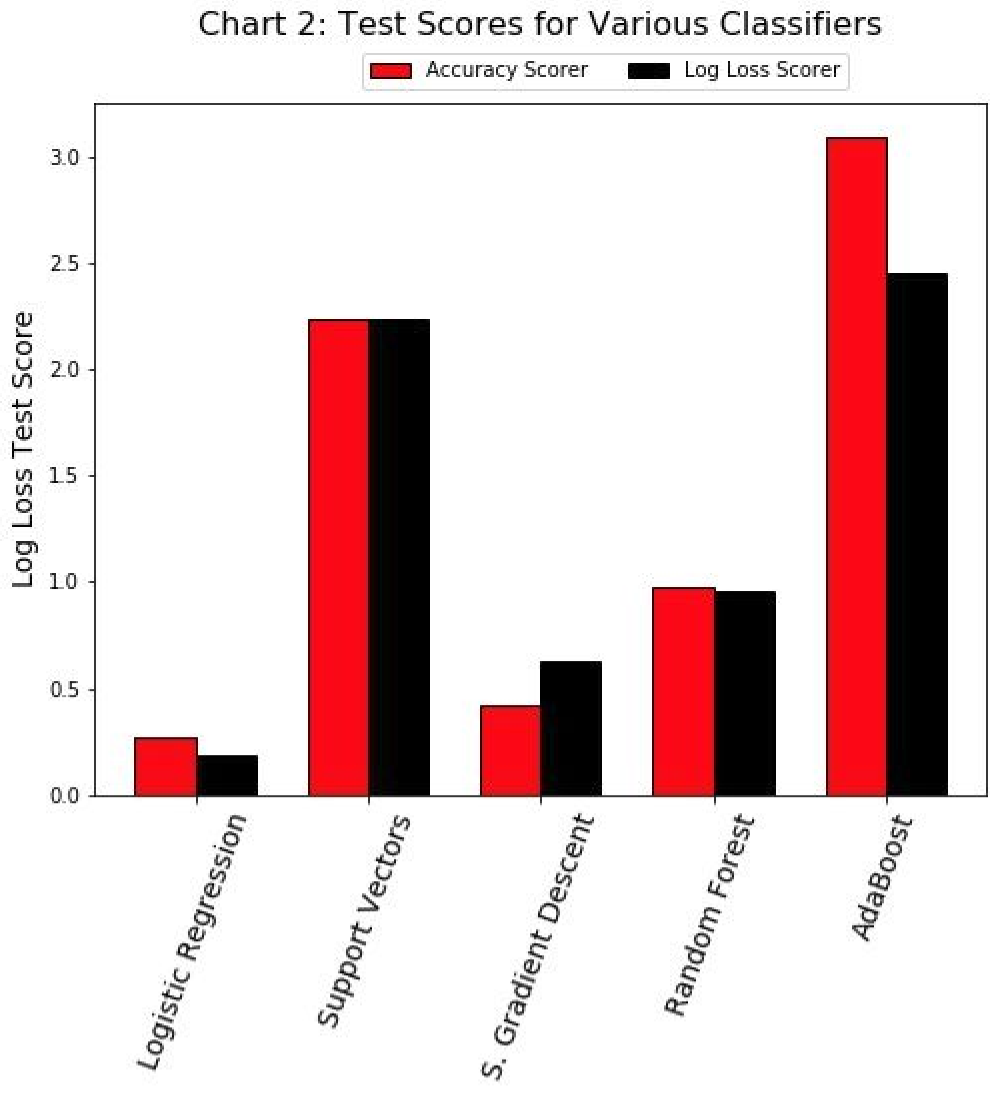
```
GRID_SEARCH_PARAM = {
    'DecisionTreeClassifier': {'min_samples_split': [2, 3, 4, 5]},
    'RandomForestClassifier': {'n_estimators': [100, 150, 200, 250],
                               'min_samples_split': [2, 3, 4, 5]},
    'AdaBoostClassifier': {'learning_rate': [0.01, 0.05, 0.1]},
    'LogisticRegression': {'C': [10.0, 12.0, 15.0, 20.0, 25.0],
                           'solver': ['liblinear', 'newton-cg', 'lbfgs']},
    'SGDClassifier': {'loss': ['hinge', 'log'],
                      'penalty': ['none', 'l2', 'l1']},
    'SVC': {'C': [40.0, 50.0, 60.0],
            'probability': [True]}
}
```

To prevent overfitting the training data, it is important to have a strategy for cross-validation. The strategy to be used is stratified 5-fold cross validation. This splits the training data into five folds, with each fold taking a turn as the validation set while the other four are used as the training set, with the results averaged. Stratification refers to the fact that each fold is balanced across the 99 classes. This is important for our data set since, without stratification of the 99 classes, some folds could contain few or none of some classes.

The results for training and validation scores of the classifiers can be found in Chart 1 below.

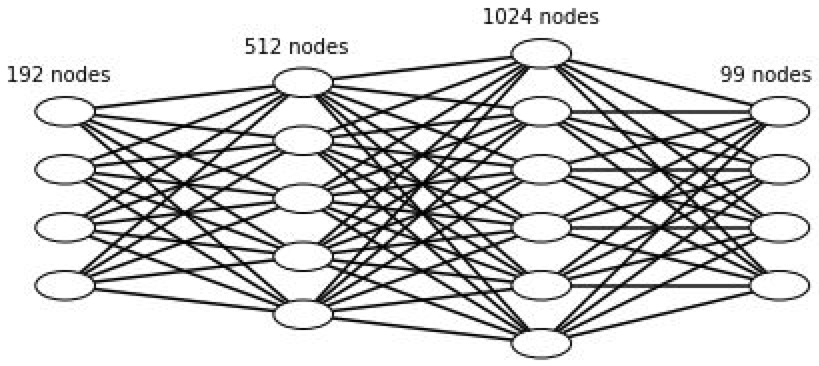


Most of the classifiers score above .94 on the validation set. Decision trees and AdaBoost do not perform well. The test scores are computed by submitting the test probabilities to Kaggle. Since Kaggle uses logarithmic loss to score its submissions, I hypothesized that using log loss as a scorer during cross-validation would increase the results on the test scores. So for each algorithm, I submitted the scores first using accuracy as a metric in cross validation and then using logarithmic loss. The results can be seen in Chart 2 below. The scores for decision trees are omitted because they scored over 12, which scaled the y-axis such that it became difficult to see the differences in the other columns.

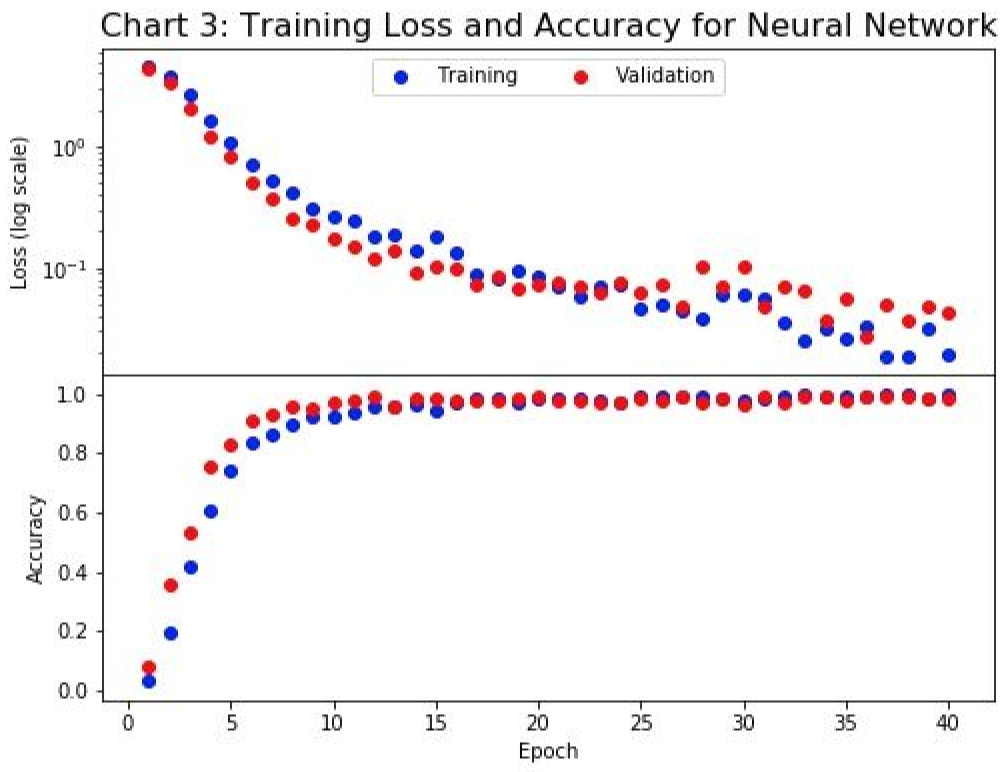


Using log loss as a scorer improved results for most of the classifiers. The best-performing algorithm is logistic regression, with a score of 0.18344. We will return to this algorithm in Step 3 for a more thorough optimization.

In Step 2, a TensorFlow neural network is trained on the data. It had two hidden layers with 512 and 1024 nodes each:



50% dropout was applied between the two hidden layers and 30% dropout was applied before the output layer. The loss is calculated as the cross entropy between the output logits and the predicted classes. Adam optimizer is used because stochastic gradient descent didn't work well. The validation set is 20% of the training data, and is randomly shuffled after every epoch. Training and validation accuracies and loss are recorded during 40 epochs, and can be seen in Chart 3.



The loss and accuracies continue to improve until about Epoch 35. The final validation accuracy is 98.5%, and the testing score is 0.05083. This score is better than any classifier from Step 1.

For Step 3 of the analysis, I attempt to improve upon the results from the first two Steps. Logistic regression was the best-performing algorithm from Step 1. In order to better tune the parameters to the dataset, several values of each parameter are provided for grid search:

```
GRID_SEARCH_PARAM = {
    'LogisticRegression': {'C': [50.0, 75.0, 100.0, 125.0, 150.0, 200.0],
                            'solver': ['newton-cg', 'lbfgs'],
                          'fit_intercept': [True, False],
                          'max_iter': [50, 100, 150, 200],
                          'tol': [1e-2, 1e-3, 1e-4, 1e-5] }
}
```

‘C’ is the inverse of the regularization strength; a high value means less regularization, and a low value means more regularization.[7] ‘solver’ refers to the possible algorithms used for optimization. ‘fit_intercept’ determines whether or not a constant is added to the function. ‘max_iter’ is the maximum number of times the algorithm is able to update without convergence. ‘tol’ is the tolerance for terminating the algorithm.

The best-performing logistic regression algorithm uses lbfgs as a solver, has a tolerance of 0.001, 100 maximum iterations, a C of 200, and has a constant value added to the function. It scores 98.5% on the validation sets and 0.13714 on logarithmic loss. This is an improvement over the Step 1 logistic regression algorithm, but not over the neural network.

As for the neural network, adding depth or more neurons with varying amounts of dropout did not improve the results.

IV. Results

Model Evaluation and Validation

The best-performing model for the problem of leaf classification was the neural network. Oddly, the results could not be improved by adding depth or regularization to the network. This is likely due to the fact that the dataset is small, with only 990 training instances. Its validation accuracy is 98.5%, which means that it can correctly classify approximately 98.5% of unseen test cases. This is quite remarkable for a multi-class problem with 99 output classes.

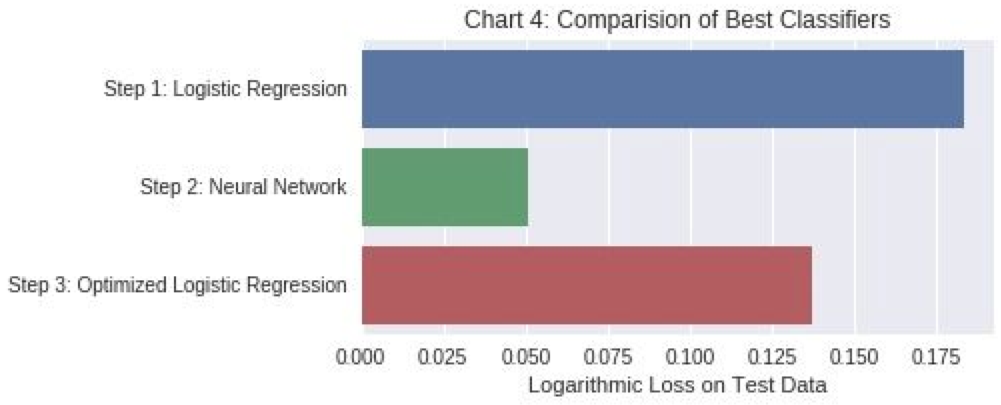
Justification

The benchmark model is a naive classifier that assumed equal probability of each species for each test instance. It scored 4.6 on logarithmic loss. Nearly all classifiers performed better than this benchmark, with the exception of the decision tree classifier.

V. Conclusion

Reflection

The goal of this project was to create a classifier for 99 species of leaves. Following a normalization of the data, Step 1 of the analysis involved training several classifiers with grid search and stratified k-fold cross validation. The best-performing algorithm was logistic regression, with logarithmic loss used to score the training data. Step 2 involved training a TensorFlow neural network on the data. Following 40 epochs, this algorithm scored 0.05 on log loss. Step 3 involved fine-tuning the algorithms from Steps 1 and 2. While the neural network could not be improved, the logistic regression algorithm was improved by a thorough grid search. The results for the best-scoring classifiers can be found in Chart 4.



This project was interesting because of its scientific applications. With more data and a custom feature extraction algorithm, it may be possible to design an algorithm that classifies every plant species on Earth. The final architecture for this project is a three-layer neural network. A general classifier, capable of classifying all species, would be much more complex. Given that the validation accuracies are above 98%, such a universal classifier may be attainable with more data and depth in training.

During each step in the analysis, validation accuracies were very close to the training accuracies. This is important because a model with a high training score with a low validation score - like the decision trees from Step 1 - suggest that the model is overfitting, and will not generalize to unseen data. This type of model is said to have high variance, and is not very robust. At the other end of the spectrum, there are models that perform poorly on both the training and validation sets - like AdaBoost from Step 1. These high bias models are not very robust either, because they are not learning any of the important structures in the data. Since the neural network and tuned logistic regression algorithm had such a small difference between training and validation scores, we know that the bias and variance are balanced to give a robust model.

The goals set in the problem statement were 99% accuracy on the validation set and under 0.1 logarithmic loss on the test dataset. The final neural network does not meet the first requirement, but does meet the second. To meet the first requirement, we could try augmenting the dataset by applying Gaussian noise to some training instances in order to make the model more robust.

Improvement

As an improvement to this program, I would do one of two things. The first is to create a custom feature generator. The solution seems somewhat lacking without a feature generator because it can not be taken into the world and applied to real leaves. The second, in the same vein as the first, is to use a convolutional neural network applied to images. In my proposal, I mentioned I wouldn't use a convolutional network because the extracted features contained more information, and a convolutional network wouldn't pick up on fine details like edge structure. By creating a dataset of lightly scanned leaves, such that the internal structure is revealed, and augmenting this dataset with rotations, a convolutional network could be very effective.

VI. References

1. Kaggle | Leaf Classification - March 29th, 2017
<https://www.kaggle.com/c/leaf-classification>

2. Kaggle | Leaf Classification | Data - March 29th, 2017
<https://www.kaggle.com/c/leaf-classification/data>

3. Exegetic Analytics | Making Sense of Logarithmic Loss - April 4th, 2017
<http://www.exegetic.biz/blog/2015/12/making-sense-logarithmic-loss/>

4. Wikipedia | Boosting (machine learning) - April 10th, 2017
<https://en.wikipedia.org/wiki/Boosting_(machine_learning)>

5. Scikit-Learn | AdaBoostClassifier - April 10th, 2017
<http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html>

6. Scikit-Learn | Stochastic Gradient Descent - April 10th, 2017
<http://scikit-learn.org/stable/modules/sgd.html>

7. Scikit-Learn | Logistic Regression - April 10th, 2017
<http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html>