CZ4041 Machine Learning

Group Report on Leaf Classification (Group 2)

Group Members	Matriculation Number	Roles
Liu Jihao Bryan	U1621401C	Coding & Report & Video
Eugene Teo	U1721688J	Coding & Report & Video
Grace Ong	U1721575H	Coding & Report & Video
Htet Naing	U1620683D	Coding & Report & Video
Leung Kai Yiu	U1522848L	Coding & Report & Video

Contents

Problem statement

Motivation

Plants play a vital role in the environment. There will be no existence of the earth's ecology without plants. However, recently, several species of plants are in danger of extinction. In order to protect plants and to catalogue various species of flora diversities, a plant database becomes very essential. Moreover, along with the conservation feature, recognition of plants has also became essential to exploit their medicinal properties and using them as alternative energy sources. There are estimated to be nearly half a million species of plants in the world. Classification of plant species has been historically problematic and often results in duplicate identifications.

Problem definition

The objective of this competition is to use binary leaf images and extracted 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. The dataset provided to us, consists of approximately 1,584 images of leaf specimens which have been converted to binary black leaves against white backgrounds. Three set of features (shape contiguous descriptor, interior texture histogram, fine-scale margin histogram) are also provided per image. Using the dataset provided, we will build a classifier that is capable of classifying and identifying plant species.

Challenges of problem

Since this is our first time attempting a Kaggle competition, it is challenging for us as we are not exposed to the process of the competition; from the process of building models from scratch to producing results for submission. There were a lot of tips given by experienced Kaggle participants in the discussion board, one such examples would be the optimal parameters for our models, which helped us immensely as it reduced the time needed to tune our models.. Another challenge we faced, was the lack of powerful hardware to train and test our models. This is especially so for image classification which is computationally expensive.

Proposed solution

Methodology

Algorithm Chain and Pipeline

1. Preprocess the data
    a. Label encoding
    b. Feature scaling
    c. Grid search with cross validation
2. Feed the pre-processed data into selected classifiers
    a. Naive Bayes
    b. Support Vector Machine
    c. K-nearest neighbours
    d. Logistic Regression
    e. Linear discriminant analysis
    f. Decision Tree
    g. Random Forest
    h. Multilayer Perceptron Neural Network
3. Evaluate the classifiers

Evaluation Metrics

- **Accuracy** - The simplest and most intuitive as it uses the ratio of correctly predicted observation to the total observation. However, accuracy is only a good measure when we have a symmetric data set, that is the values of false positive and false negatives have the same cost.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

- **Log loss** - The formula for multiclass logarithmic loss is as such:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij}),$$

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 **pij** is the predicted probability that observation i belongs to class j. In order to avoid the extremes of the multiclass logarithmic loss function, predicted probabilities are replaced with: max(min(p,1−10−15),10−15).

Data Pre-processing

We then proceed with the encoding of categorical and text features, feature scaling and grid search with cross validation.

Label Encoding

Numerical data is required for training the models in our project. However the plant species column are of categorical or text data. Hence, we use label encoding to convert them into numbers so that we can provide meaningful information to train the model.

```
le = LabelEncoder().fit(train_raw.species) # Instantiate a
LabelEncoder and fit to the given label data
labels = le.transform(train_raw.species)  # encode species strings
and return labels with value between 0 and n_classes-1
```

Figure 1: Implementation of Label Encoding

Feature Scaling

Different features have non-standardised weight. Therefore, we weigh all features equally to ensure the features have a standardised feature value and prevent any dominance in dimensions. We tried multiple scaling methods including the standard scalar and min-max scaler. However, we find that standard scaling works the best as we will get consistent mean-centering and unit scale differences, this will allow us to have a distance measure that contribute equally across all features.

$$\boxed{z = \frac{x - \mu}{\sigma}} \qquad \boxed{\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)} \qquad \boxed{\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}}$$

This standard scaling (z-normalisation) formula takes the x value and minus the mean and divide by the standard deviation.

```
# Standardize the training data.
    scaler = StandardScaler().fit(X_train)
    X_train_scaled = scaler.transform(X_train)
    scaler = StandardScaler().fit(X_test)
    X_test_scaled = scaler.transform(X_test)
```

Figure 2: Implementation of Feature Scaling using z-Normalisation

Grid Search with Cross Validation

We use grid search with cross validation to perform hyperparameter optimisation to find the most optimal parameters for our model. In our case, we perform grid search with cross validation on the support vector machine as well as the logistic regression model and just cross validation on the k-nearest neighbor.

- **Grid Search** - Most commonly used method to find the values of the important parameters of a model. It basically means trying all possible combinations of the parameters of interest also known as hyperparameter optimisation.

```
    param_grid = {'C': [ 1000, 10000],
                  'tol': [0.000001, 0.00001]}
    log_reg = LogisticRegression(solver='newton-cg',
multi_class='multinomial')
    grid_search = GridSearchCV(log_reg, param_grid,
scoring='neg_log_loss', refit='True', n_jobs=1, cv=ss_split)
    grid_search.fit(X_train_scaled, y_train)
```

Figure 3: Example of using Grid Search for Logistic Regression implementation

- **Cross Validation** - A statistical method of evaluating generalization performance that is more stable and thorough than using only one test set. In cross-validation, the data is instead split repeatedly and multiple models are trained. For this project, we use the stratified shuffle split strategy. The strategy will first split the data into k parts. After this step, it will pick one part to use as test set, then it repeat the process k-1 times to get k-1 test set. Because it shuffle each time before it split, the test set can overlap. This introduce randomness and low bias in the subsampling.

```
# construct the iterator
ss_split = StratifiedShuffleSplit(n_splits=10, test_size=0.2,
random_state=0)
ss_split.get_n_splits(train, labels)
```

Figure 4: Implementation of Cross Validation using Stratified Shuffle Split Strategy

Classifiers

Naive Bayes

Naive Bayes is a probabilistic model that is based on the Bayes theorem. The Bayes theorem describes the probability of an object based on prior knowledge of conditions that might be related to that object. We use Naive Bayes as it is the easiest and fastest in predicting the class of the test set.

```
def run_naive_bayes(train, test, ss_split, labels):
   # prepare training and test data
   X_train, X_test, y_train, y_test = hpr.prepData(train, test,
ss_split, labels);

   clf = GaussianNB().fit(X_train, y_train) # Instantiate a
classifier and fit this classifier to the data
   print ('ML Model: Naive Bayes')
   # Cross-validation
   scores = cross_val_score(GaussianNB(), train.values, labels,
cv=ss_split)
   print ('Mean Cross-validation scores: {}'.format(np.mean(scores)))
   # Accuracy
   train_predictions = clf.predict(X_test)
   acc = accuracy_score(y_test, train_predictions)
   # Logloss
   train_predictions_p = clf.predict_proba(X_test)
   ll = log_loss(y_test, train_predictions_p)

   test_predictions = clf.predict_proba(test)
   return test_predictions, acc, ll
```

Figure 5: Implementation of Naive Bayes Classifier

Support Vector Machine

Support Vector Machine (SVM) is a discriminative classifier defined by a separating hyperplane differentiating the two classes. The SVM algorithm finds the hyperplane that gives the largest minimum distance to the training data samples. The unknown samples will be classified under the class that gives the optimal separating hyperplane which maximizes the margin of the training data.

```
def run_support_vector_machine(train, test, ss_split, labels):
   # prepare training and test data
   X_train, X_test, y_train, y_test = hpr.prepData(train, test,
ss_split, labels);
   clf = SVC(probability=True)

   # Gird search
   param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
                 'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}

   grid_search = GridSearchCV(SVC(probability=True),
param_grid=param_grid, cv=ss_split)
   grid_search.fit(X_train, y_train)

   print ('Best parameter: {}'.format(grid_search.best_params_))
   print ('Best cross-validation accuracy score:
{}'.format(grid_search.best_score_))
   print ('\nBest
estimator:\n{}'.format(grid_search.best_estimator_))
   print ('ML Model: Suppoort Vector Machine')
   # Accuracy
   train_predictions = grid_search.predict(X_test)
   acc = accuracy_score(y_test, train_predictions)
   # Logloss
   train_predictions_p = grid_search.predict_proba(X_test)
   ll = log_loss(y_test, train_predictions_p)

   test_predictions = grid_search.predict_proba(test)
   return test_predictions, acc, ll
```

Figure 6: Implementation of Support Vector Machine

Logistic Regression

The Logistic Regression is a predictive analysis that is used to describe data and explain the relationship between one dependent binary variable and one or more nominal independent variables. Logistic regression outputs the probabilities of a specific class and those probabilities can be converted into class predictions.

```
def run_logistic_regression(train, test, ss_split, labels):
   # prepare training and test data
   X_train, X_test, y_train, y_test = hpr.prepData(train, test,
ss_split, labels);

   # Standardize the training data.
   scaler = StandardScaler().fit(X_train)
   X_train_scaled = scaler.transform(X_train)
   scaler = StandardScaler().fit(X_test)
   X_test_scaled = scaler.transform(X_test)

   param_grid = {'C': [ 1000, 10000],
                 'tol': [0.000001, 0.00001]}
   log_reg = LogisticRegression(solver='newton-cg',
multi_class='multinomial')
   grid_search = GridSearchCV(log_reg, param_grid,
scoring='neg_log_loss', refit='True', n_jobs=1, cv=ss_split)
   grid_search.fit(X_train_scaled, y_train)

   print ('Best parameter: {}'.format(grid_search.best_params_))
   print ('Best cross-validation neg_log_loss score:
{}'.format(grid_search.best_score_))
   print ('\nBest
estimator:\n{}'.format(grid_search.best_estimator_))
   print ('ML Model: Logistic Regression')
   # Accuracy
   train_predictions = grid_search.predict(X_test_scaled)
   acc = accuracy_score(y_test, train_predictions)
   # Logloss
```

```
    train_predictions_p = grid_search.predict_proba(X_test_scaled)
    ll = log_loss(y_test, train_predictions_p)

    scaler = StandardScaler().fit(test)
    test_scaled = scaler.transform(test)
    test_predictions = grid_search.predict_proba(test_scaled)

    return test_predictions, acc, ll
```

Figure 7: Implementation of Logistic Regression

Linear Discriminant Analysis

Linear discriminant analysis is a dimensionality reduction technique used to find a linear combination of features that characterizes two or more classes of object. Its goal is to reduce the dimensions by removing the redundant and dependent features by transforming the features from higher dimensional space to a space with lower dimensions.

```
def run_linear_discriminant_analysis(train, test, ss_split, labels):
    # prepare training and test data
    X_train, X_test, y_train, y_test = hpr.prepData(train, test, 
ss_split, labels);

    clf = LinearDiscriminantAnalysis().fit(X_train, y_train)
    print ('ML Model: Linear Discriminant Analysis')

    train_predictions = clf.predict(X_test)
    acc = accuracy_score(y_test, train_predictions)

    train_predictions_p = clf.predict_proba(X_test)
    ll = log_loss(y_test, train_predictions_p)

    test_predictions = clf.predict_proba(test)
    return test_predictions, acc, ll
```

Figure 8: Implementation of Linear Discriminant Analysis

K-Nearest Neighbours

The k-nearest neighbours is a non-parametric and lazy learning method used for classification. The input consists of k closest training examples in the feature space. The output is a class membership. An object is classified by a plurality vote of its neighbors with the object being assigned to the class most common among its k nearest neighbours.

```
def run_k_nearest_neighbours(train, test, ss_split, labels):
   # prepare training and test data
   X_train, X_test, y_train, y_test = hpr.prepData(train, test, 
ss_split, labels);

   clf = KNeighborsClassifier(3)  # Instantiate a classifier
   clf.fit(X_train, y_train) # Fit this classifier to the data
   print ('ML Model: K-Nearest Neighbours')

   # Cross-validation
   scores = cross_val_score(KNeighborsClassifier(3), train.values, 
labels, cv=ss_split)

   # Accuracy
   train_predictions = clf.predict(X_test)
   acc = accuracy_score(y_test, train_predictions)
   # Logloss
   train_predictions_p = clf.predict_proba(X_test)
   ll = log_loss(y_test, train_predictions_p)

   test_predictions = clf.predict_proba(test)
   return test_predictions, acc, ll
```

Figure 9: Implementation of K-Nearest Neighbours

Decision Tree

Decision Tree algorithm uses a set of test questions and if-else conditions in a tree structure. The decision rules become more complex when the tree becomes deeper. Decision Tree is used for both classification and regression models. It breaks down a dataset into small subsets represented by smaller trees subsets and simultaneously incrementally developing an associated decision tree. The resulting tree is a tree with decision nodes and leaves. Also, the branches of the tree will show the factors within the analysis that are considered relevant to the decision.

```
def run_decision_tree(train, test, ss_split, labels):
    # prepare training and test data
    X_train, X_test, y_train, y_test = hpr.prepData(train, test,
ss_split, labels);

    clf = DecisionTreeClassifier().fit(X_train, y_train)
    print ('ML Model: Decision Tree')

    # Accuracy
    train_predictions = clf.predict(X_test)
    acc = accuracy_score(y_test, train_predictions)
    # Logloss
    train_predictions_p = clf.predict_proba(X_test)
    ll = log_loss(y_test, train_predictions_p)

    test_predictions = clf.predict_proba(test)
    return test_predictions, acc, ll
```

Figure 10: Implementation of Decision Tree

Random Forest

Random Forest is also another ensemble classifier used in our project. It creates a set of decision trees from randomly selected subset of training tree set. It then aggregates the votes from different decision trees to decide the final class of the test object. This method is an improved version of decision trees, as it handles overfitting and error due to bias, better. Accuracy predictions are made using the average prediction values of all trees from the test data set x' shown below.

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x')$$

```python
def run_random_forest(train, test, ss_split, labels):
    # prepare training and test data
    X_train, X_test, y_train, y_test = hpr.prepData(train, test, 
ss_split, labels);

    clf = RandomForestClassifier().fit(X_train, y_train)
    print ('ML Model: Random Forest')

    # Accuracy
    train_predictions = clf.predict(X_test)
    acc = accuracy_score(y_test, train_predictions)
    # Logloss
    train_predictions_p = clf.predict_proba(X_test)
    ll = log_loss(y_test, train_predictions_p)

    test_predictions = clf.predict_proba(test)
    return test_predictions, acc, ll
```

Figure 11: Implementation of Random Forest

Multilayer Perceptron Neural Network

Multilayer Perceptron is a feedforward artificial neural network. In a neural network, there is a set of interconnected layers. The layers consist of: an input layer, hidden layers and an output layer. Each node in the layer has an activation function, in order to do complex and nonlinear function mapping. Each edge in a layer will have a weight, which will be learned through training to optimise the model loss.

```python
def run_mlp_neural_network(train, test, ss_split, labels):

    # prepare training and test data
    X_train, X_test, y_train, y_test = hpr.prepData(train, test, 
ss_split, labels);

    scaler = StandardScaler().fit(X_train)
    X_train_scaled = scaler.transform(X_train)
    scaler = StandardScaler().fit(X_test)
    X_test_scaled = scaler.transform(X_test)


    print ('ML Model: MLP Neural Network')
    model =
MLPClassifier(hidden_layer_sizes=(150,),activation='logistic',solver=
'lbfgs',alpha=0.001,max_iter=200,early_stopping=True,validation_fract
ion=0.2,
learning_rate='adaptive',tol=1e-8,random_state=1).fit(X_train_scaled,
y_train)
    # Accuracy
    train_predictions = model.predict(X_test_scaled)
    acc = accuracy_score(y_test, train_predictions)
    # Logloss
    train_predictions_p = model.predict_proba(X_test_scaled)
    ll = log_loss(y_test, train_predictions_p)

    scaler = StandardScaler().fit(test)
    test_scaled = scaler.transform(test)
    test_predictions = model.predict_proba(test_scaled)
    return test_predictions, acc, ll
```

Figure 12: Implementation of Multilayer Perceptron

Data Post-processing

After obtaining prediction results for the given test data, the results are formatted and then exported as “submission.csv” file. The file is later used for submission to Kaggle.

```
# Format DataFrame
submission = pd.DataFrame(test_predictions, columns=classes)
submission.insert(0, 'id', test_ids)
submission.reset_index()

# Export Submission
submission.to_csv('submission.csv', index = False)

# Double check the output
submission.head()
```

Figure 13: Data formatting for submission to Kaggle

Experiments

Results

The given data for training machine learning models consists of 990 samples. The sample data is then divided into two sets of data, the training data and the test data. The training data set consists of 80 percent of the given data and the test data set has 20 percent of the given data.

Given training data samples	990 (100%)
Training data set	792 (80%)
Test data set	198 (20%)

After that, the accuracy percentage and the log loss value for the test data set are calculated to observe the performance of each machine learning model. The best model will then be used to generate final submission file (in csv format) for leaderboard score evaluation on Kaggle.

ML Model	**Accuracy**	**Log Loss**
Naive Bayes	55.0505%	15.5066
Support Vector Machine	93.4343%	2.30096
Logistic Regression	98.9899%	0.02539
Linear Discriminant Analysis	97.4747%	0.92252
K-Nearest Neighbours	87.8788%	2.25021
Decision Tree	68.6869%	10.8152
Random Forest	87.8788%	1.49840
Multilayer Perceptron Neural Network	**100%**	**0.01090**

Figure 14: Experiment Results showing accuracy and log loss values for various models

According to the experiment result table above, Multilayer Perceptron Neural Network (MLP) outperforms the rest of the models. Hence, the prediction results generated by using MLP will be used to upload to Kaggle for the leaderboard score evaluation.

ANALYSIS

Naive Bayes

Naive Bayes aims to take advantage of using small training set due to its being high bias and low variance. At the same time, it converges quicker than discriminative models. However, it is not able to learn interactions between features and it assumes independence, which is rarely true. In our experiment, Naive Bayes predicts extremely fast, but the log loss is too high.

Support Vector Machine

SVM first transforms input data into a higher dimensional space via a kernel function. Optimal Separating Hyperplane (OSH) is constructed between the two classes in the the transformed space. In our experiment, the feature vector extracted from leaf's contour is transformed by SVM. After that, it constructs OSH by maximizing the margin between the classes [1]. The benefit of using SVM for classification tasks is twofold. SVM not only gives a simple geometric interpretation but also gives a sparse solution. The advantage of SVM over neural networks is that the dimensionality of the input space does not affect the computational complexity of SVMs. The drawback of SVM is that it uses the large number of support vectors from the training set when carrying out classification tasks.[2] Another disadvantage is its slow training in learning the model.

Logistic Regression

Logistic regression measures the relationship between the dependent variable (our label, what we want to predict) and the one or more independent variables (our features), by estimating probabilities using its underlying logistic function. Since logistic regression is binary classifier and this is a multiclass classification task, it automatically uses the one-versus-one strategy. This has a big advantage as it will only need to train it on part of the training set for the 2 classes it distinguishes between. This is faster than to training just one at a large dataset. It performed fairly well for the leaf classification case, with one of the highest accuracy. There is however still a log loss, which may be due to the fact that for each feature logistic regression separates the image into 2 region by a linear boundary, one for each class. The data might not be fully linearly separable which would result in some losses. [3]

Linear Discriminant Analysis

Linear Discriminant Analysis is used as a dimensionality reduction technique. It uses multidimensional datasets that have multiple variables that have correlations to each other and plots it in two or three dimensions in order to identify trends and classify the data. However, Linear Discriminant Analysis assumes the data is Gaussian with each variable is shaped like a Bell Curve when plotted, and each attribute has the same variance. This may not be true in the given data set, making the model not learn perfectly, resulting in a higher log loss than required.

K-Nearest Neighbours

KNN is a non-parametric and lazy learning algorithm which is measured based on the k nearest neighbors of each query point [4]. After much testing for different values for K (from 1 to 25), our model picked 3 entries that are closest to the query point and classified the query point based on the most common label within the entries. Eventually, we achieved the maximum accuracy 87.8788% with 2.25021 log loss. This undesired result may be due to the small K that leads to more complex model, resulting in overfitting.

Decision Tree

Decision Tree determines which features of the leaves and under what conditions to use to split the Decision Tree in order to classify the leaves. Proper pruning of the tree is needed to make the tree perform better by disregarding features of low importance to increase accuracy. However, Decision Tree can tend to create over-complicated trees that do not generalize data well, making it overfit and lead to low prediction accuracy. Decision Trees can also be unstable and small changes in the dataset can lead to massively different trees, sampling the data differently might lead to higher accuracy but could also lead to lower accuracy, making optimizing difficult.

Random Forest

Random forests builds decision trees using random data samples and selects the best solution based on the prediction from each tree through voting [5]. After much testing with different parameters, our model used the default parameters (100 trees in the forest, gini impurity, no max depth, etc) and achieved the maximum accuracy 87.8788% with 1.49840 log loss. Since Random Forest is an improved version of Decision Tree, the performance has become much better as compared to that of Decision Tree. However, it is still not as good as the best result that have been achieved so far (i.e. Logistic Regression).

Multilayer Perceptron Neural Network

The goal of MLP is to do a functional mapping from the input to the output. MLP performs well with both small and large datasets. In our experiment, as for training our MLP model, "Limited-memory BFGS (L-BFGS)" is used for weight optimization since it can converge faster and perform better for small datasets. The L-BFGS is a quasi-Newton method which approximates BFGS. It is commonly used to approximate parameters in machine learning while maintaining the limited amount of computer memory usage [6].

Only one hidden layer with 150 neurons is used to train our MLP model as it is aimed to be able to generalise well while avoiding overfitting. MLP trains iteratively since at each time step, the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters. In our hidden layer, the logistic sigmoid activation function is also used. "The advantage of this activation function is, unlike linear function, the output of the activation function is always going to be in range (0,1) compared to (-inf, inf) of linear function" [7].The downside of MLP would be that it is prone to overfitting due to the high number of parameters within the network itself. To alleviate this issue, the regularization term *alpha* is added to the loss function that shrinks model parameters to prevent overfitting.

Overall, compared to other machine learning methods, using MLP for the leaf classification is not only fast in execution but also efficient in recognition.

Evaluation score and ranked position on Kaggle

Evaluation score

For this topic, “Leaf Classification”, the final score for the submission is evaluated based on the multi-class logarithmic loss. It is calculated using the formula below:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij}),$$

Where :

N is the number of images in the test set (N = 594 according to the given data)

M is the the number of species labels (M = 99 according to the given data)

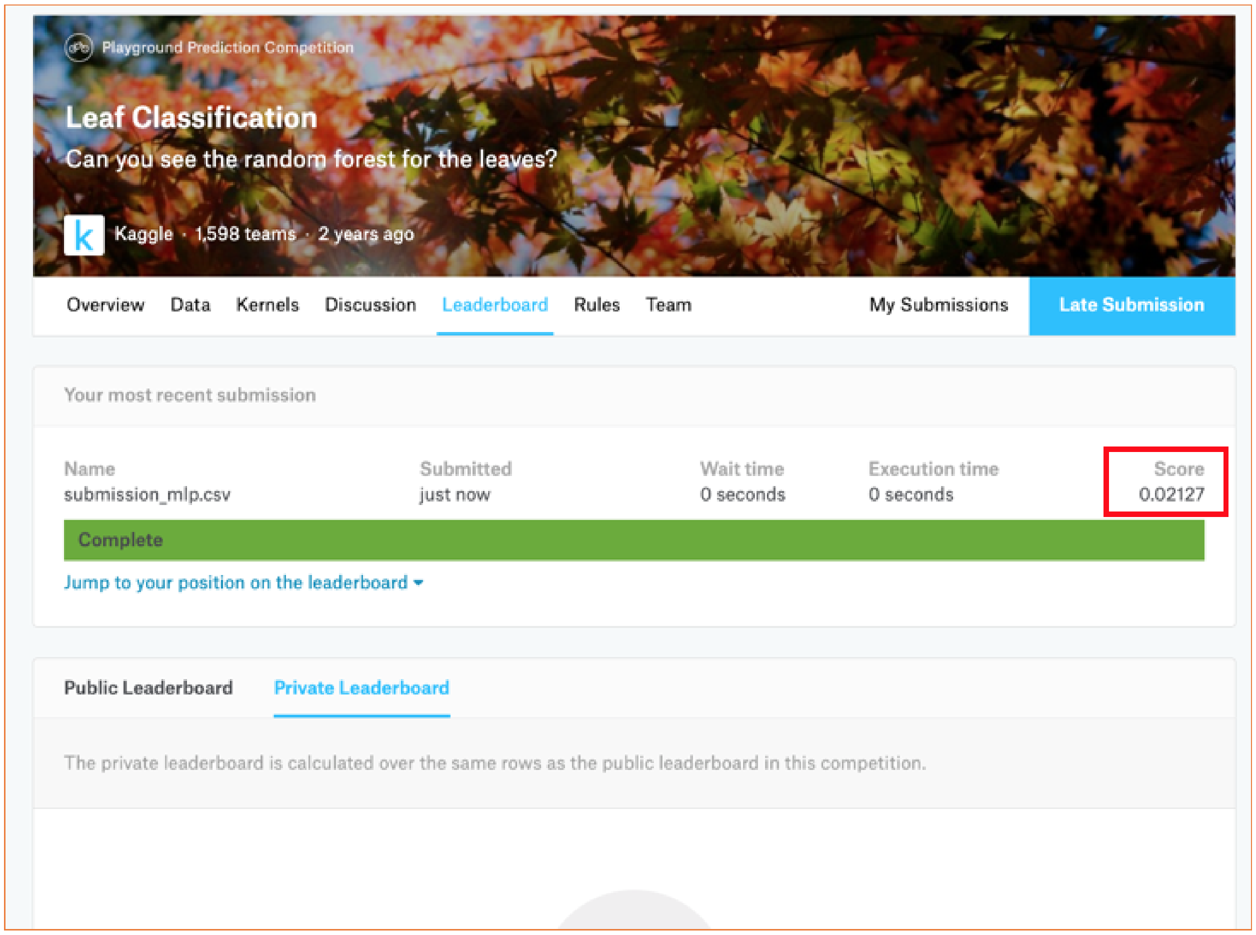
log is the natural logarithm

yij is 1 is observation *i* is in class *j* and 0 otherwise

pij is the predicted probability that observation *i* belongs to class *j*

Therefore, the lesser the final score, the better the result since the score represents the value of the multiclass logarithmic loss. According to the experiment result table shown in Figure 14, since MLP gives the best result among all models that are experimented for leaf classification, the given test data that consists of 594 samples is fed to our trained MLP model. Finally, the submission file generated by using our trained MLP model on the test data is then uploaded to Kaggle for the leaderboard score evaluation.

The final score on Kaggle is evaluated based on the multi-class logarithmic loss. We have achieved a very promising score of **0.02127** as indicated by the red rectangle in the following figure.


Playground Prediction Competition

Leaf Classification

Can you see the random forest for the leaves?

Kaggle · 1,598 teams · 2 years ago

Overview Data Kernels Discussion Leaderboard Rules Team My Submissions Late Submission

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission_mlp.csv	just now	0 seconds	0 seconds	0.02127

Complete

Jump to your position on the leaderboard

Public Leaderboard Private Leaderboard

The private leaderboard is calculated over the same rows as the public leaderboard in this competition.

Figure 15: Evaluation score after submitting the prediction results for the given test data

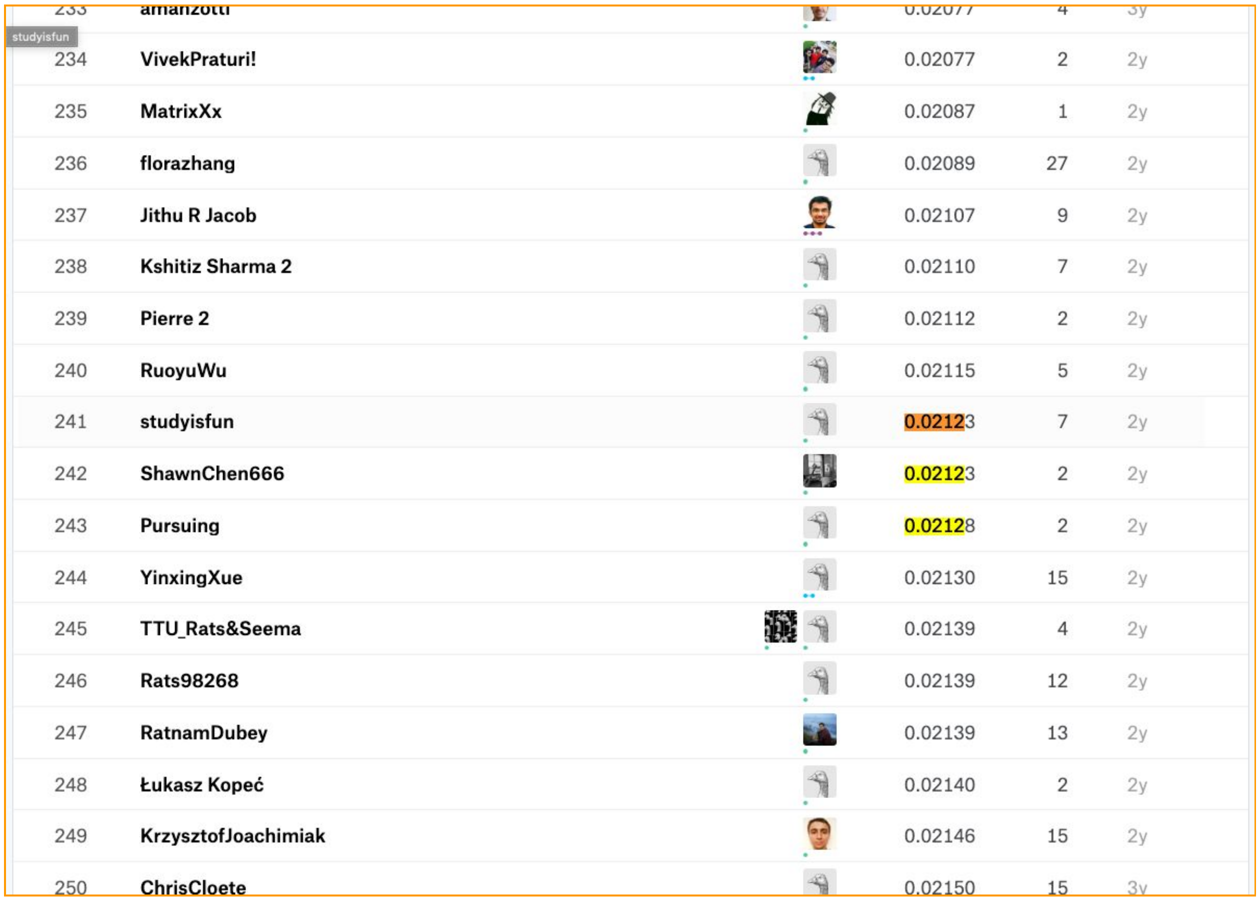
Ranking position

Public Leaderboard

There are total of **1598** teams on the Public Leaderboard. Since we have achieved the final score of **0.02127**, the result will place us on the **243th** position on the leaderboard as highlighted in the figure below. Thus,

$$(243 / 1598) \times 100\% = 15.2\%$$

We have secured the top 15% of the leaderboard based on our final evaluation score.



233	amanzotti	0.02077	4	3y
234	VivekPraturi!	0.02077	2	2y
235	MatrixXx	0.02087	1	2y
236	florazhang	0.02089	27	2y
237	Jithu R Jacob	0.02107	9	2y
238	Kshitiz Sharma 2	0.02110	7	2y
239	Pierre 2	0.02112	2	2y
240	RuoyuWu	0.02115	5	2y
241	studyisfun	0.02123	7	2y
242	ShawnChen666	0.02123	2	2y
243	Pursuing	0.02128	2	2y
244	YinxingXue	0.02130	15	2y
245	TTU_Rats&Seema	0.02139	4	2y
246	Rats98268	0.02139	12	2y
247	RatnamDubey	0.02139	13	2y
248	Łukasz Kopeć	0.02140	2	2y
249	KrzysztofJoachimiak	0.02146	15	2y
250	ChrisCloete	0.02150	15	3y

Figure 16: Possible placement on the Public Leaderboard (out of 1598 teams)

Summary

From study of above classification techniques we have come up with following conclusion.

As for Naive Bayes Classifier, it only requires a small amount of training data to estimate the necessary parameters. It predicts extremely fast compared to more sophisticated methods. However, it is also known to be weak at estimating as proven in our experiment [8].

Support Vector Machine (SVM) is good at generalizing data and robust even when the training sample has some bias. The bottlenecks of SVM are its slow training to learn the model and difficulty to understand its algorithmic structure.

While logistic regression is both simple and powerful linear classification algorithm, it becomes unstable either when the label classes are well separated or when only few training samples are available to estimate the parameters. [9]

Linear Discriminant Analysis (LDA) tackles the issues of Logistic Regression It is the appropriate linear method for multi-class classification problems. The drawback of LDA is overfitting and it requires methodical validation or testing. It is of good practice to experiment both logistic regression and linear discriminant analysis for classification tasks. [10]

K Nearest Neighbours Classifier (kNN) is one of the simplest machine learning methods for classification tasks and also robust in terms of search space but it is sensitive to noisy or irrelevant inputs.

Decision Tree is simple to understand and requires minimal data preprocessing. It is capable of handling both numerical and categorical data. Its downside is that it can be unstable since small variations in the data, can lead to generation of a completely different tree [11].

The strength of Random Forest lies in reduction of over-fitting and also performs better than decision trees in terms of accuracy in most cases. However, it works like a black-box making its users have less control over what the model does.

Generally, it is well-known that neural networks can outperform nearly every other machine learning algorithms. Therefore, in our experiment, Multilayer Perceptron Neural Network (MLP) shows the best performance both in terms of accuracy and multi-class logarithmic loss. Artificial Neural networks including MLP suffer from poor interpretability of their results due to their black-box nature and they are also computationally expensive. [12]

To conclude, “no free lunch theorem” states that there is no perfect machine learning algorithms that will perform well at any problem. Given a problem, a particular method is suitable and able to perform better than another. Therefore, it is important to understand how each machine learning algorithm work and consider their pros and cons before applying them in a specific problem domain.

References

[1] M. Kumar, M. Kamble, S.Pawar, P. Patil, N. Bonde, “Survey on Techniques for Plant Leaf Classification ”, International Journal of Modern Engineering Research (IJMER), Vol.1, Issue.2, pp-538-544. [Online]. Available: https://pdfs.semanticscholar.org/2acc/1cf77c71b98c4ea19e00fdb6dbd2caf3688b.pdf

[2] M. Kumar, M. Kamble, S.Pawar, P. Patil, N. Bonde, “Survey on Techniques for Plant Leaf Classification ”, International Journal of Modern Engineering Research (IJMER), Vol.1, Issue.2, pp-538-544. [Online]. Available: https://pdfs.semanticscholar.org/2acc/1cf77c71b98c4ea19e00fdb6dbd2caf3688b.pdf

[3] machinelearning-blog.com (2018), “ The Logistic Regression Algorithm”. [Online]. Available: https://machinelearning-blog.com/2018/04/23/logistic-regression-101/

[4] A. Navlani (2018), “KNN Classification using Scikit-learn”. [Online]. Available: https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

[5] A. Navlani (2018)**, “**Understanding Random Forests Classifiers in Python”. [Online]. Available: https://www.datacamp.com/community/tutorials/random-forests-classifier-python

[6] D.R.S. Saputroa and P. Widyaningsihb, “Limited Memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) Method for The Parameter Estimation on Geographically Weighted Ordinal Logistic Regression Model (GWOLR)”, (AIP Conference Proceedings 1868, 040009, 2017). [Online]. Available: https://aip.scitation.org/doi/pdf/10.1063/1.4995124?class=pdf

[7] Avinash S.V. (2017), “Understanding Activation Functions in Neural Networks”. [Online]. Available: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

[8] G. Gahukar (2018), “Classification Algorithms in Machine Learning”. [Online]. Available: https://medium.com/datadriveninvestor/classification-algorithms-in-machine-learning-85c0ab65ff4

[9] Jason. B. (2016), “Linear Discriminant Analysis for Machine Learning”. [Online]. Available: https://machinelearningmastery.com/linear-discriminant-analysis-for-machine-learning/

[10] Jason. B. (2016), “Linear Discriminant Analysis for Machine Learning”. [Online]. Available: https://machinelearningmastery.com/linear-discriminant-analysis-for-machine-learning/

[11] G. Gahukar (2018), “Classification Algorithms in Machine Learning”. [Online]. Available: https://medium.com/datadriveninvestor/classification-algorithms-in-machine-learning-85c0ab65ff4

[12] N. Donges (2018), “Pros and Cons of Neural Networks”. [Online]. Available: https://towardsdatascience.com/hype-disadvantages-of-neural-networks-6af04904ba5b