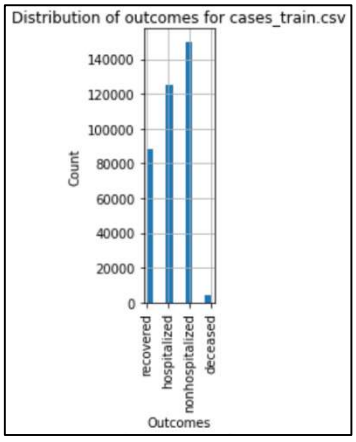
Project Title

Training and Evaluating Models to Classify Health Outcomes on Covid-19 Dataset.

Problem Statement

The goal of this project is to predict the health outcome of an individual who has been infected with Covid-19. The method involves using the provided Covid-19 datasets and two types of classification models: Random Forest and LightGBM. The potential outcomes that the classifiers can predict are `hospitalized`, `nonhospitalized`, `deceased`, and `recovered`. These outcomes represent the resulting health status of an infected person. Specifically, maximizing the F1-score of correctly classifying `deceased` cases is of interest for this problem, as the `deceased` case is the most severe possible outcome.

Figure 1



Dataset Description and EDA

Skewed Distribution of Outcomes

The following datasets were provided: `cases_train`, `cases_test`, and `location`. `Cases_train` contains over 367,000 rows of datapoints. However, it should be noted that the distribution of outcomes over these datapoints is skewed. Per the Figure 1, the number of deceased cases is only 4,499, which translates to 1.2% of the dataset.

Heatmap for Training and Testing Datasets

Heatmaps showing the geographic location of the data points in the `cases_train` and `cases_test` datasets can be seen in Figures 2 and 3. Visually, they show that the two datasets are fairly representative of each other since they both cover a similar set of geographic locations.

Figure 2

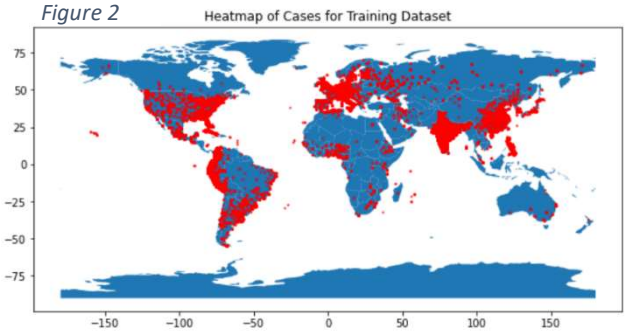
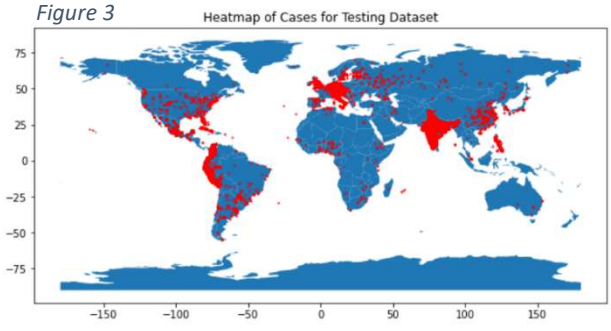


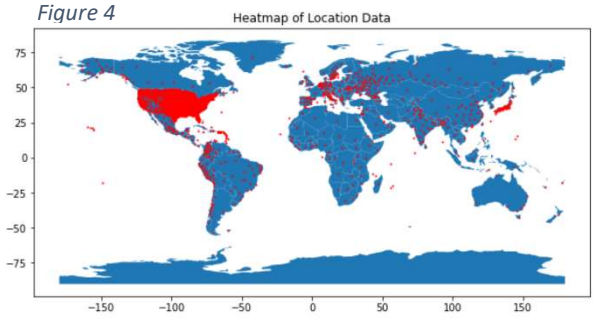
Figure 3



Heatmap for Location Dataset

The heatmap of the `location` dataset was also plotted to see what areas the data covered (see Figure 4). As shown, most of the points are in the United States, which is accounted for by the data aggregation, transformation, and joining steps that are done on the United States data during the Data Preparation steps.

Figure 4



Missing Values for Training Dataset

The `cases_train` dataset has missing values for all columns except for the outcome. Refer to Figure 5 to see the count of all missing values. As shown in the figure, `additional_information` is the column with the most missing values (344,912) followed by `age` (209,265) and `sex` (207,084).

Figure 5: Missing Values

age	209265
sex	207084
province	4106
country	18
latitude	2
longitude	2
date_confirmation	288
additional_information	344912
source	128478
outcome	0

Data Preparation

Data Cleaning and Imputing Missing Values

Missing Value	**Why & how the missing values were filled**
Age	This column had over 200,000 rows of missing data in the dataset. As age is a critical factor to determine the outcome of Covid-19 patients, it was important to fill missing age values. The missing values were determined by creating a record of all the average ages for each given province and then referencing the province of each missing age to impute the missing value. If the average age for a province was unavailable, the average age of a country was used, and if neither was available, the median of all the existing ages was used to fill missing values.
Sex	Similar to age, this column also had over 200,000 rows of missing values. In the training dataset, the outcome column was used to determine the percentage of males and females who had `recovered`, `hospitalized`, etc. as their outcome. For each potential outcome, the sex that had a bigger percentage of receiving that outcome was used to fill in the missing values. Because the test dataset had no outcome column, a dictionary containing the larger gender majority percentage for each province or country generated from the training dataset was used to fill missing values.

Province	There were over 4,000 missing values for the province which contained valuable information for determining if the location was a contributing factor to Covid-19 outcomes. Geopy was used to compute the missing provinces by inputting the given latitude/longitudes.
Country	The 18 missing values for the country column were filled to have a more detailed and complete dataset for Milestone 2. Similar to the province, the country was determined using geopy by inputting the latitude and longitude of each missing data row to determine the missing country.
Date Confirmation	The 300 missing values for the date confirmation were filled in to have a full and complete dataset for Milestone 2. After clearing the odd formats for the dates, the empty cells for date confirmation were filled by the most occurring date out of all the existing date confirmations.

Dealing with Outliers

There were two attribute groups that contained outliers. The first attribute is `age` because some ages reached ranges way above the average life expectancy. These numerical inconsistencies were automatically detected using a normal distribution and setting z-index boundaries. The possible age ranges were plotted in the table `./plots/Age_plot.png` which revealed a normal distribution in the data as displayed by the bell curve plotted in `./plots/Age_normal.png`. A z-index range from -4 to 4 was determined to be the bounds such that any ages with z-indexes that resided outside of the bounds were determined to be outliers. For this dataset, ages that were greater than or equal to 100 were classified to be outliers.

Similarly, some latitude and longitude attributes were considered outliers because after imputing missing province data, some remaining missing values remained due to geopy not recognizing the coordinates that point to undefined locations. Due to the improbable case that patients from an undefined region were tested for Covid-19, latitudes/longitudes that lead to lingering missing provinces were considered outliers and removed.

Transformation on US Data in Locations Dataset

US rows were filtered and grouped by `Province_State`. The following aggregations were used:

Column	**Aggregation Method**
Last_Update	The mode of all rows belonging to the state. This would make the aggregated Last_Update value reflect the most frequently seen date.
Lat / Long_	The average of Lat and Long_ values was used. Averaging these would bring the aggregated coordinate to roughly the middle of all the individual locations.
Confirmed, Deaths,	The sum aggregation was used for these columns. It makes sense to count all values amongst a state's various counties/areas to get a total for the state.

Recovered, Active	
Combined_Key	The mode was used with the first value (i.e. the county/area) removed. This made the Combined_Key value only show '<state>, US'.
Incidence_Rate	The incidence rate represents cases per 100,000 people, so the value is already normalized to have a consistent base (100,000). Therefore, the values of the individual counties/areas were averaged to get a representative rate for the state.
Case-Fatality_Ratio	Since this ratio is the number of deaths / number of cases, a weighted average of the state's counties' ratios was used. Taking Alabama for example, its Case-Fatality_Ratio = Σ[(Case-Fatality_Ratio for county) * (Confirmed # for county) / (Confirmed # for Alabama)] for all of Alabama's counties. Using the weighted sum ensured that the ratio was correctly weighted by the number of confirmed cases per county.

Joining the Cases and Location Dataset

The `province, country` key was used to join the cases datasets to the location dataset. This strategy was used because it was easy to implement and check matching values after imputing the missing provinces/states.

There were notably some mismatches between the datasets; for example, `Puerto Rico` was noted as a country in the cases dataset but as a province/state in the locations dataset. An exploratory pass of the datasets was done to find these mismatches and helper functions were created to align them and create a `Combined_Key` for the cases datasets.

Note there were some rows (~8%) with NaN values for countries such as Peru and Russia because they had no matching 'Province_States' in the location dataset. The values for these rows were filled with the mean values of the aggregation of the countries in `location.csv`.

Oversampling

The training dataset and problem statement fits the definition of an imbalanced classification problem [1]. The dataset is skewed so that the `deceased` outcome is the minority class, representing only 1.2% of all cases. Considering the dataset contains both numerical and categorical features, SMOTENC was used to oversample the deceased cases. In order to determine a good balance, 20% of the training dataset was held out and used as validation to evaluate the effectiveness of oversampling. The result was that increasing the number of deceased cases in the training set to 37,500 (i.e. approximately 10% of the dataset) yielded the best results. It was found that excessively oversampling led to the models excessively misclassifying other outcomes as `deceased`.

Classification Models

Some columns were dropped from the dataset. `Combined_Key`, `latitude`, `longitude`, `Province_State`, `Country_Region`, `Lat`, and `Long_` were removed because they held duplicated data that was already included in the columns `province_filled` and `country_filled`. Other columns such as `additional_information`, `source`, and `Last_Update` were removed because they held metadata that was irrelevant to the classification task.

LightGBM was chosen to be the boosting model. Gradient boosting was chosen instead of adaptive boosting because AdaBoost is not optimized for speed or handling noisy data. Since a significant number of values were imputed (200,000+ imputations for missing `age` values alone), it is possible that the Covid-19 dataset is noisy and would not be handled well by AdaBoost. Gradient boosting is also more flexible than AdaBoost since more hyperparameters can be tuned to improve performance. Of the gradient boosters, LightGBM was chosen over XGBoost because it can handle categorical data (such as the `country` and `state` columns in the dataset) efficiently without converting them with one-hot encoding.

Random Forest was chosen to be the second model. By producing multiple different possible outputs from the multiple decision trees that are generated, Random Forest is a highly accurate classifier that can produce great results even without hyperparameter tuning. In terms of accuracy, Random Forest outperforms other faster classifiers such as Decision Trees, Naive Bayes, Logistic regression, KNN, etc. However, more accurate classifiers such as MLP, Kernel SVM, and Random Forest all suffer from the problem of being incredibly slow and consuming a lot of processing power. In the end, the Random Forest classifier was chosen among these accurate classifiers due to its flexibility in handling numerical/categorical data and being robust when handling outliers, noise, and missing values. Considering that the dataset given in the project is composed of both numerical and categorical type data, classifiers such as Linear regression or Linear SVMs would have a difficult time working with this dataset. Additionally, the speed of Random Forest is not the worst as very accurate classifiers that can outperform Random Forest, such as MLP, are usually slower and consume more resources.

Initial Evaluation and Overfitting

Initial Evaluation

The baseline models for Random Forest and LightGBM produced the following results:

LightGBM									
Train	precision	recall	f1-score	support		**Validation**	precision	recall	f1-score	support
deceased	0.72	0.04	0.08	3640		deceased	0.64	0.04	0.08	859
hospitalized	0.97	0.99	0.98	99847		hospitalized	0.97	0.99	0.98	25153
nonhospitalized	0.98	1	0.99	120040		nonhospitalized	0.98	1	0.99	29929
recovered	0.98	0.96	0.97	70555		recovered	0.98	0.96	0.97	17580

Random Forest									
Train	precision	recall	f1-score	support	**Validation**	precision	recall	f1-score	support
deceased	0.88	0.01	0.02	3640	deceased	0.83	0.01	0.01	859
hospitalized	0.96	0.95	0.96	99847	hospitalized	0.97	0.95	0.96	25153
nonhospitalized	0.98	1	0.99	120040	nonhospitalized	0.98	1	0.99	29929
recovered	0.92	0.96	0.94	70555	recovered	0.92	0.96	0.94	17580

As seen in the F1-score column for both the train and validation sets, the models performed very well at correctly classifying cases that were `hospitalized`, `non-hospitalized`, and `recovered`. However, they performed poorly for classifying `deceased` cases. Confusion matrices were generated (see the four figures saved in the `/plots/confusion_matrices_before_rebalancing` directory) to see what was happening with the misclassifications. The matrices showed that the majority of `deceased` cases were being misclassified as `hospitalized`. The misclassification could be because the `deceased` and `hospitalized` outcomes are similar in that they are the two most severe of the possible outcomes. It could also be because there are fewer `deceased` cases so the models did not sufficiently weigh them during training. After doing this baseline approach, the 'deceased' data points were oversampled in the dataset in order to reduce the imbalance. The oversampling improved the rate at which `deceased` cases were misclassified as `hospitalized`. The updated confusion matrices that demonstrate the improvement can be found in the `/plots/confusion_matrices_after_rebalancing` directory.

Overfitting

The two baseline approaches were analyzed for overfitting by varying one hyperparameter each. The maximum depth for LightGBM was varied between 2 and 18 and the accuracy scores were plotted to see if the model was overfitting (see `/plots/overfitting_check_gdb.png`). The accuracy for the validation set did not decrease as the maximum depth increased, which would have indicated overfitting. Therefore, the LightGBM model did not overfit, but the accuracy did plateau for maximum depth >= 8. It is also interesting to note that the accuracy score for the validation set was consistently higher than that of the train set. This could be due to the split of the dataset, since the train set had a higher percentage of `deceased` outcomes than the validation set (by chance), and it is the `deceased` cases that are often misclassified.

Similarly, the maximum depth for Random Forest was set to be between 10 and 100 and the accuracy scores were plotted to see if the model was overfitting (see `/plots/overfitting_check_rf.png`). By nature, Random Forest is designed to be less prone to overfitting as they use random creations of forests to reduce overfitting within decision trees. Therefore, an interesting result was that the accuracy for the training set remained the same while the validation set fluctuated between the values of 0.959 and 0.960 but never went into a steady decrease which would indicate overfitting. This means that the Random Forest model is not overfitting because the accuracy score stabilizes near the first few trees as more trees are added to the model. Similar to LightGBM, the accuracy score for the validation set was always higher than the train set which could also be correlated to the chance that more `deceased` outcomes were in the training set than the validation set.

Hyperparameter Tuning

GridSearchCV was used to tune the hyperparameters for the LightGBM model. Due to the high efficiency of LightGBM, GridSearchCV could be run in a reasonable amount of time even with a large combination of hyperparameters. The cross validation value was set to 3 and the hyperparameters that were tuned were `learning_rate`, `max_depth`, and `num_leaves`. The `learning_rate` was varied between 0.01 and 0.11 to explore a wide range. As for `max_depth`, the overfitting test done earlier showed that the value of 8 was an interesting pivot point, so `max_depth` was varied between 6 and 12 to straddle that value. The hyperparameter `num_leaves` was varied between 25 and 45 so that values above and below the model's default value (31 per documentation [2]) would be tested.

RandomizedSearchCV was used to tune the hyperparameters for the Random Forest model because the large amounts of time required to train a single Random Forest model made it impractical to test all the possible combinations of a large variety of hyperparameters using GridSearchCV. Therefore, in order to find the best set of hyperparameters but also explore a variety of hyperparameters more efficiently, RandomizedSearchCV was used because of its ability to select a specified amount of random combinations to explore, which made the hyperparameter tuning process much faster.

For the Random Forest model, the cross-validation value was set to 5 and the hyperparameters that were fine tuned were `n_estimators`, `max_depth`, and `min_samples_split`. Additionally, only 20 random hyperparameter combinations were chosen each time to speed up the tuning process and consequently allow more time to experiment with different hyperparameter options. For the hyper parameter values chosen, it was determined during the previous overfitting section that the training accuracy begins to flatten when `n_estimators` is equal to a range around 30 to 110. Thus, the `n_estimators` were set to the same range for hyperparameter tuning. The `min_samples_leaf` was set to a range of 2 to 4 and `max_depth` was set to a range of 50 to 100 to determine if increasing these parameters will further improve the performance of the model, as the default values for `min_samples_left` is 1 and `max_depth` is None [3].

Results

The `micro` average was used for calculating the overall recall instead of the `macro` and `weighted` average because the `micro` average is the only one that considers the imbalanced number of outcome labels. It is interesting to note that using the `micro` average makes the overall recall score identical to the overall accuracy because of the class imbalance consideration.

LightGBM Best Hyperparameters

The following is an excerpt of the results yielded from hyperparameter tuning LightGBM:

combination	f1_deceased	recall_deceased	overall_accuracy	overall_recall
{'learning_rate': 0.01, 'max_depth': 6, 'num_leaves': 25}	0.333812331	0.20784	0.897683127	0.897683127
{'learning_rate': 0.01, 'max_depth': 6, 'num_leaves': 30}	0.334192916	0.207946667	0.897850839	0.897850839
{'learning_rate': 0.01, 'max_depth': 6, 'num_leaves': 35}	0.334041807	0.207733333	0.89807344	0.89807344
{'learning_rate': 0.01, 'max_depth': 6, 'num_leaves': 40}	0.33452338	0.208053333	0.898164919	0.898164919
{'learning_rate': 0.01, 'max_depth': 6, 'num_leaves': 45}	0.334599886	0.208106667	0.898167969	0.898167969
{'learning_rate': 0.01, 'max_depth': 8, 'num_leaves': 25}	0.376176518	0.243573333	0.900393972	0.900393972
{'learning_rate': 0.01, 'max_depth': 8, 'num_leaves': 30}	0.382594927	0.24888	0.901144105	0.901144105

The full dataset can be found in `/results/lightgbm_gridsearch_results.csv`. The GridSearchCV algorithm determined that the best set of parameters are: `learning_rate`: 0.11, `max_depth`: 12, `num_leaves`: 45.

Random Forest Best Hyperparameters

The following is an excerpt of the results yielded from hyperparameter tuning Random Forest:

combination	f1_deceased	recall_deceased	overall_accuracy	overall_recall
{'n_estimators': 80, 'min_samples_split': 2, 'min_samples_leaf': 4, 'max_features': 'log2', 'max_depth': 100}	0.401948277	0.258385714	0.840637341	0.840637341
{'n_estimators': 40, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'log2', 'max_depth': 90}	0.524597613	0.385885714	0.858004916	0.858004916
{'n_estimators': 80, 'min_samples_split': 2, 'min_samples_leaf': 4, 'max_features': 'log2', 'max_depth': 90}	0.400699884	0.257485714	0.840412618	0.840412618
{'n_estimators': 40, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'log2', 'max_depth': 70}	0.518967282	0.380485714	0.856836897	0.856836897
{'n_estimators': 70, 'min_samples_split': 2, 'min_samples_leaf': 3, 'max_features': 'log2', 'max_depth': 80}	0.449871567	0.305828571	0.846912981	0.846912981

The full dataset can be found in `/results/randomforest_randomsearch_results.csv`. The algorithm eventually determined that the best combination was: `n_estimators`: 40, `min_samples_leaf`: 2, `max_depth`: 90.

Conclusion

Comparative Study

After hyperparameter tuning, the LightGBM and the Random Forest models were both used to predict the outcomes of the oversampled training set and the validation set to compare and evaluate their performance. The following results are shown below:

LightGBM

Train	precision	recall	f1-score	support	Validation	precision	recall	f1-score	support
deceased	0.84	0.35	0.5	37500	deceased	0.23	0.21	0.22	859
hospitalized	0.8	0.98	0.88	99847	hospitalized	0.97	0.98	0.98	25153
nonhospitalized	1	0.99	0.99	120040	nonhospitalized	0.99	0.99	0.99	29929
recovered	0.99	0.98	0.98	70555	recovered	0.99	0.98	0.98	17580
overall accuracy				0.91	overall accuracy				0.98

Random Forest

Train	precision	recall	f1-score	support	Validation	precision	recall	f1-score	support
deceased	0.85	0.3	0.44	37500	deceased	0.15	0.19	0.17	859
hospitalized	0.67	0.98	0.88	99847	hospitalized	0.97	0.98	0.98	25153
nonhospitalized	0.98	0.99	0.99	120040	nonhospitalized	0.99	0.99	0.99	29929
recovered	0.98	0.96	0.97	70555	recovered	0.99	0.96	0.98	17580
overall accuracy				0.85	overall accuracy				0.97

It is interesting to note that the F1-score for the validation set is significantly lower than the F1-score on the training set for both models. The biggest source of the drop comes from the precision scores. The low precision indicates that the models are incorrectly classifying non-deceased outcomes as deceased, possibly because the oversampled training dataset provides higher chances to correctly identify `deceased` outcomes, whereas there is less room for error with the validation set.

The tables demonstrate that the LightGBM model outperformed the Random Forest model for overall accuracy. In terms of predicting the `deceased` outcome label on the oversampled training set and the validation set, the LightGBM model generally performed better than the Random Forest model in standard metrics such as precision, recall, and F1-score. Even though Random Forest scored slightly higher than LightGBM for precision on the training set (0.84 vs 0.85), LightGBM still had a higher F1-score because of its superior recall score (0.35 vs. 0.3). In terms of the validation set, the LightGBM model had higher overall accuracy (0.98 vs 0.97) and higher precision (0.23 vs 0.15), recall (0.21 vs 0.19), and F1-score (0.22 vs 0.17) for `deceased` cases.

Advantages and Disadvantages of LightGBM

Aside from the performance metrics, an advantage of LightGBM was that it was computationally efficient. It could run all the hyperparameters given to the model within an hour. Part of the efficiency came from the model's ability to handle categorical data without using one-hot encoding, which also simplified the set-up process. Another aspect of the model that could be considered both a positive and a negative was the plethora of potential hyperparameters that could be tuned. Having all the options that LightGBM provides allows greater control of the model, but also adds a layer of complexity when determining which hyperparameters to use.

Advantages and Disadvantages of Random Forest

As described before, a big advantage of Random Forest is that they are prone to avoid overfitting due to the generation of multiple trees. Therefore, choosing hyperparameters that are too large would not necessarily harm the performance of the model which makes Random Forest one of the more accurate and powerful classifiers. However, a major disadvantage to Random Forest is the large amounts of processing time required to generate a series of trees. This makes the process of hyperparameter tuning very difficult because finding an optimal set of hyperparameters among a large number of combinations incurs a prohibitively large running time due to the generation of multiple models. This means that it could be very likely that the most optimal combination of hyperparameters could be missed.

Model Choice

LightGBM was picked to predict the test dataset because the model consistently outperformed Random Forest when comparing the `deceased` F1-scores and overall accuracy.

Prediction on Test Dataset

After determining that LightGBM yields a consistently higher F1 score on `deceased` outcomes, the full training dataset was used to train the model. The training dataset was oversampled using

SMOTENC and the model was trained with the hyperparameters that were determined during hyperparameter tuning. The resulting predictions can be found in `/results/predictions.txt`.

Lessons Learnt and Future Work

A big lesson that was learned is the importance of a balanced dataset when training machine learning models. If a dataset is heavily imbalanced like the current dataset on Covid-19, predicting the outcome for minority labels accurately becomes significantly harder. On the surface, a training classifier may seem to have a high accuracy score when predicting these outcome labels, but it is also important to investigate further metrics such as F1-score and precision on each outcome label to identify the true weaknesses of a model.

Another lesson from this project is that data modeling while tuning multiple hyperparameters takes a long time. In the future, more of the processing could be moved to Google Colab to take advantage of the CPU and GPU accessible there. By using the computing resources available on Google Colab, more models and potentially even neural networks could be trained and evaluated as classifiers for the Covid-19 dataset.

In future work, additional preprocessing steps could be explored, such as binning for age and dates and combining over and undersampling to further correct the imbalanced dataset.

Contributions

The separate contributions of each member were as follows:

Lucia	**Josh**
- Exploratory Data Analysis and outcome labels - Transformation and joining datasets - Building and evaluating the LightGBM model - Checking LightGBM model for overfitting - Hyperparameter tuning for LightGBM model	- Data cleaning and splitting the dataset - Imputing missing values and handling outliers - Building and evaluating the RF model - Checking RF model for overfitting - Hyperparameter tuning for RF model

In addition to the tasks above, both members worked together to write and edit each of the Milestone reports and to ensure the submission packages were complete.

References

[1] https://machinelearningmastery.com/what-is-imbalanced-classification/

[2] https://lightgbm.readthedocs.io/en/latest/Parameters.html

[3] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html