Mushrooms Classification
Machine Learning Engineer Nanodegree

Capstone Project

Reem Khaled
November 25, 2018

I. Definition

Project Overview

A mushroom, or toadstool, is the fleshy, spore-bearing fruiting body of a fungus, typically produced above ground on soil or on its food source[1]. Mushrooms are used widely in cooking recipes because they are tasty and because all types of edible mushrooms contain varying degrees of protein and fibre. They also contain B vitamins as well as a powerful antioxidant called selenium, which helps to support the immune system and prevent damage to cells and tissues.

The problem is there are many mushroom species that are not edible and can in fact cause stomach pains or vomiting if eaten, and in some cases could be fatal, such as the common death cap mushroom[2]. People identify poisonous mushrooms by their color, shape and other features. There are many rules and guidelines to follow when picking good mushrooms, such as avoiding mushrooms with red on the cap or stem, avoiding mushrooms with white spore print, and most importantly, not consuming any mushrooms unless being 100% sure of what they are. The dataset used for this project is “Mushroom Data Set” from “UCI Machine Learning Repository”[3]. It was published in 1987 and it has 8124 rows and 23 columns, and all its attributes have categorical values that discrbe different aspects of mushrooms, like 'cap-color', 'stalk-shape' and others. The dataset was also introduced in Kaggle.[4]

Problem Statement

Mushrooms can be classified as edible (fit to be eaten) or poisonous. Eating poisonous mushrooms may cause stomach pains and other problems, or even may be fatal. Each mushroom should be observed separately to collect certain pieces of information such as the cap color, stalk shape and others. Given such information, we can identify edible mushrooms from poisonous ones.

Objective:

To classify mushrooms into two classes: edible or poisonous, using a machine learning model that can predict the category of each mushroom based on its features (information).

Machine learning algorithms used to solve this problem:

- **Logistic Regression:** a widely used algorithm if the dependent variable is dichotomous (binary). It is used to describe data and to explain the relationship between the binary target variable and one or more other variables.[5]

- **Naive Bayes:** is a classification technique based on Bayes' Theorem with an assumption of independence among predictors.[6]

- **Random Forest:** is an ensemble classifier that creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object.[7]

- **Support Vector Machine:** is an algorithm that outputs an optimal hyperplane that seperates the plotted datapoints.[8]

The dataset will be preprocessed before using machine learning models. The main two steps for preprocessing are using One-Hot-Encoding (creating dummy variables) technique to convert categorical variables into binary variables becuse the dataset contains categorical variables, and searching for and dealing with missing variables.Some other steps like removing unnecessary columns, seperating the target variable and splitting the dataset into training and testing sets will be used.

Metrics

There are two main performance measures that are used for this dataset: testing accuracy and type II error[9] (false negatives). Testing accuracy will ensure that future observations will be classified correctly. Considering the "poisonous" class as positive and "edible" as negative, type II error in our case is classifying poisonous mushrooms as edible, which is the worst case in our scenario. The chosen model is the model with the highest test accuracy and with the least type II error. The following metrics was recorded for both training and testing sets: F1-score, precision and recall, along with time needed to train each model and time it takes to predict new data points class. Now we will discuss each performance metric independently, please note that:

TP: true positive: the mushroom is poisonous and our model classified it as poisonouns.

TN: true negative: the mushroom is edible and our model classified it as edible.

FP: False positive: the mushroom is edible but our model classified it as poisonouns.

FN: False negative: the mushroom is poisonous but our model classified it as edible.

- **Accuracy**[10]

Accuracy is the most widely used performance metric in binary classification problems. It the the number of correctly classified data points over the total number of observation. By using accuracy in training, we can enusre that our model uses the data in the best possible way (understands the complexity of the data and doesn't undefit), and when using accuracy in testing set we ensure that the model generalizes well (doesn't overfit). We will also use K-fold cross validation with the accuracy metric to ensure that our results are corrct as much as possible.

$$Accuracy = \frac{Number\, of\, Correct\, Classifications}{Total\, Number\, of\, Classfications\, Made} = \frac{TP+TN}{TP+TN+FP+FN}$$

- **Type II Error**

It the FN which is the number of data points classified as negative while they are positive in the reality. In our case, it is classifying a mushroom as edible while it is poisonous.

$$Type\, II\, Error = Number\, of\, False\, Negatives$$

- **Recall (Sensitivity)**[11]

It is a statistical measure of the performance of a binary classification test. It measures the proportion of actual positives that are correctly identified as such. It is also called true positive rate.

$$Sensitivity = \frac{TP}{TP+FN}$$

- **Precision (Specificity)**[11]

It is a statistical measure of the performance of a binary classification test. It measures the proportion of actual negatives that are correctly identified as such. It is also called true negative rate.

$$Specificity = \frac{TN}{TN+FP}$$

- **F1-score (F-measure)**[12]

It is a measure of a test's accuracy, it considers both the precision p and the recall r of the test to compute the score. The F1-score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

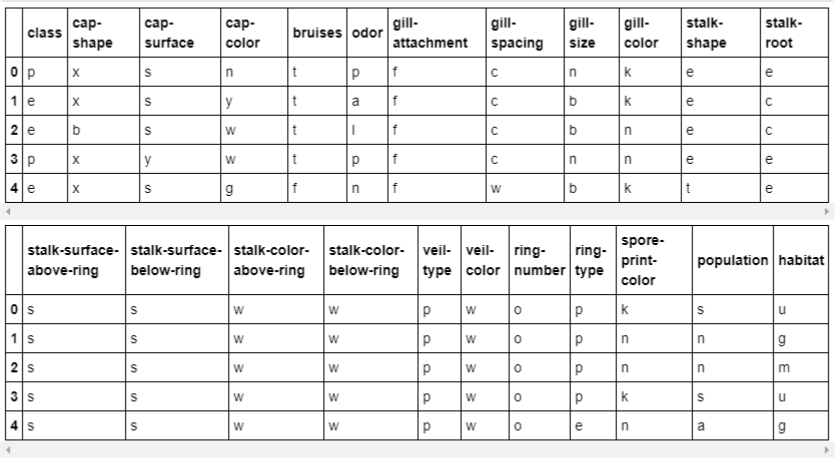
- **Time**

We measured each model training time (time it takes to fit the data), and the testing time which is prediction time (time needed to predict new instances).

II. Analysis

Data Exploration

The dataset used for this project is “Mushroom Data Set” from “UCI Machine Learning Repository”[3]. It is textual file in ’.csv’ format (comma seperated varaibles). It was published in 1987 and it has 8124 rows and 23 columns. Lets take a look at first 5 rows in the dataset:



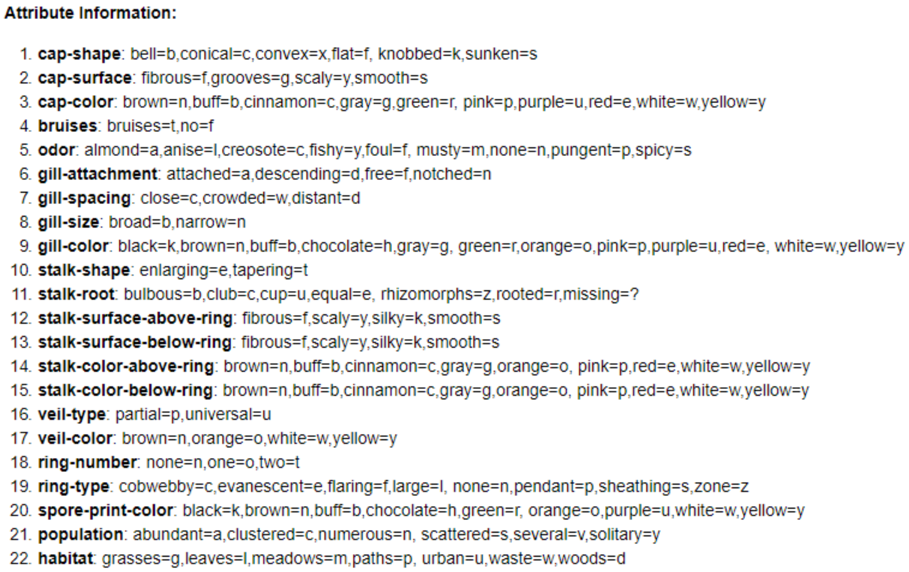
	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	stalk-shape	stalk-root
0	p	x	s	n	t	p	f	c	n	k	e	e
1	e	x	s	y	t	a	f	c	b	k	e	c
2	e	b	s	w	t	l	f	c	b	n	e	c
3	p	x	y	w	t	p	f	c	n	n	e	e
4	e	x	s	g	f	n	f	w	b	k	t	e

	stalk-surface-above-ring	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population	habitat
0	s	s	w	w	p	w	o	p	k	s	u
1	s	s	w	w	p	w	o	p	n	n	g
2	s	s	w	w	p	w	o	p	n	n	m
3	s	s	w	w	p	w	o	p	k	s	u
4	s	s	w	w	p	w	o	e	n	a	g

Figure 1: First 5 Rows in The Dataset Displayed in Two Tables

Each row represents a data point: one mushroom observation, each column specifies a specific aspect of a mushroom (a piece of information). By looking at the first 5 rows we can see that columns contain information about the shape of the mushroom, its cap, stalk, surface, odor and others. Also, all the columns contain categorical values. Our target variable is the ’class’ column (first column) which represents the class of each mushroom (’e’ as edible and ’p’ as poisonous). Other columns are our features that will be fed to the machine learning models.

Here is a detailed information about each column possible values:



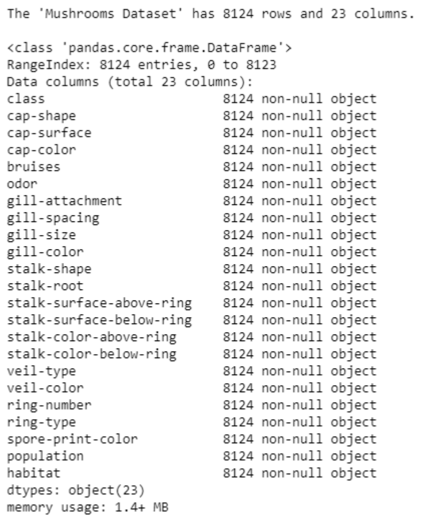
Attribute Information:

1. **cap-shape**: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
2. **cap-surface**: fibrous=f,grooves=g,scaly=y,smooth=s
3. **cap-color**: brown=n,buff=b,cinnamon=c,gray=g,green=r, pink=p,purple=u,red=e,white=w,yellow=y
4. **bruises**: bruises=t,no=f
5. **odor**: almond=a,anise=l,creosote=c,fishy=y,foul=f, musty=m,none=n,pungent=p,spicy=s
6. **gill-attachment**: attached=a,descending=d,free=f,notched=n
7. **gill-spacing**: close=c,crowded=w,distant=d
8. **gill-size**: broad=b,narrow=n
9. **gill-color**: black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e, white=w,yellow=y
10. **stalk-shape**: enlarging=e,tapering=t
11. **stalk-root**: bulbous=b,club=c,cup=u,equal=e, rhizomorphs=z,rooted=r,missing=?
12. **stalk-surface-above-ring**: fibrous=f,scaly=y,silky=k,smooth=s
13. **stalk-surface-below-ring**: fibrous=f,scaly=y,silky=k,smooth=s
14. **stalk-color-above-ring**: brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
15. **stalk-color-below-ring**: brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
16. **veil-type**: partial=p,universal=u
17. **veil-color**: brown=n,orange=o,white=w,yellow=y
18. **ring-number**: none=n,one=o,two=t
19. **ring-type**: cobwebby=c,evanescent=e,flaring=f,large=l, none=n,pendant=p,sheathing=s,zone=z
20. **spore-print-color**: black=k,brown=n,buff=b,chocolate=h,green=r, orange=o,purple=u,white=w,yellow=y
21. **population**: abundant=a,clustered=c,numerous=n, scattered=s,several=v,solitary=y
22. **habitat**: grasses=g,leaves=l,meadows=m,paths=p, urban=u,waste=w,woods=d

Figure 2: Attribute Information

Examining each attribute value set (possibe values) is the best way to understand the data. Notice the question mark symbol in the 'stalk-root' column: '?', which represents missing values.

Another check for dataset information:



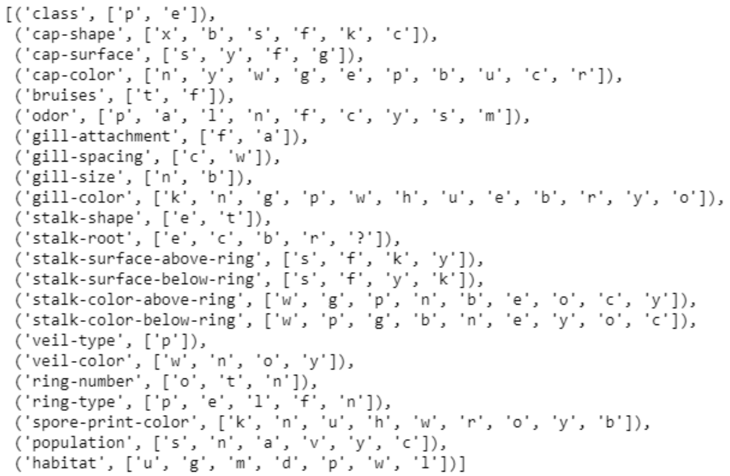
```
The 'Mushrooms Dataset' has 8124 rows and 23 columns.

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 23 columns):
class                       8124 non-null object
cap-shape                   8124 non-null object
cap-surface                 8124 non-null object
cap-color                   8124 non-null object
bruises                     8124 non-null object
odor                        8124 non-null object
gill-attachment             8124 non-null object
gill-spacing                8124 non-null object
gill-size                   8124 non-null object
gill-color                  8124 non-null object
stalk-shape                 8124 non-null object
stalk-root                  8124 non-null object
stalk-surface-above-ring    8124 non-null object
stalk-surface-below-ring    8124 non-null object
stalk-color-above-ring      8124 non-null object
stalk-color-below-ring      8124 non-null object
veil-type                   8124 non-null object
veil-color                  8124 non-null object
ring-number                 8124 non-null object
ring-type                   8124 non-null object
spore-print-color           8124 non-null object
population                  8124 non-null object
habitat                     8124 non-null object
dtypes: object(23)
memory usage: 1.4+ MB
```

Figure 3: Dataset Information: No. of Rows, Columns, Columns Names and Their Data Types

From the first glance, the dataset doesn't appear to have any null values, but as we noticed before we found that 'stalk-root' column contains null values but they are filled with '?'.

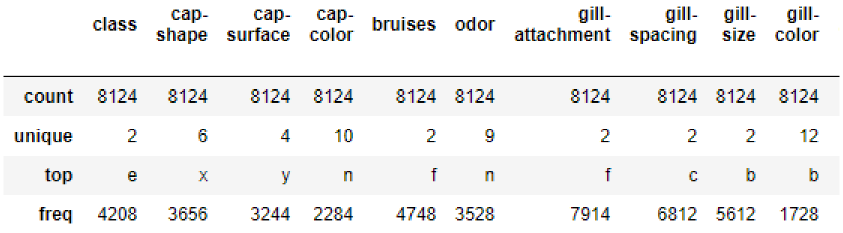
Checking unique values of the dataset:



```
[('class', ['p', 'e']),
 ('cap-shape', ['x', 'b', 's', 'f', 'k', 'c']),
 ('cap-surface', ['s', 'y', 'f', 'g']),
 ('cap-color', ['n', 'y', 'w', 'g', 'e', 'p', 'b', 'u', 'c', 'r']),
 ('bruises', ['t', 'f']),
 ('odor', ['p', 'a', 'l', 'n', 'f', 'c', 'y', 's', 'm']),
 ('gill-attachment', ['f', 'a']),
 ('gill-spacing', ['c', 'w']),
 ('gill-size', ['n', 'b']),
 ('gill-color', ['k', 'n', 'g', 'p', 'w', 'h', 'u', 'e', 'b', 'r', 'y', 'o']),
 ('stalk-shape', ['e', 't']),
 ('stalk-root', ['e', 'c', 'b', 'r', '?']),
 ('stalk-surface-above-ring', ['s', 'f', 'k', 'y']),
 ('stalk-surface-below-ring', ['s', 'f', 'y', 'k']),
 ('stalk-color-above-ring', ['w', 'g', 'p', 'n', 'b', 'e', 'o', 'c', 'y']),
 ('stalk-color-below-ring', ['w', 'p', 'g', 'b', 'n', 'e', 'y', 'o', 'c']),
 ('veil-type', ['p']),
 ('veil-color', ['w', 'n', 'o', 'y']),
 ('ring-number', ['o', 't', 'n']),
 ('ring-type', ['p', 'e', 'l', 'f', 'n']),
 ('spore-print-color', ['k', 'n', 'u', 'h', 'w', 'r', 'o', 'y', 'b']),
 ('population', ['s', 'n', 'a', 'v', 'y', 'c']),
 ('habitat', ['u', 'g', 'm', 'd', 'p', 'w', 'l'])]
```

Figure 4: Dataset Information: No. of Rows, Columns, Columns Names and Their Data Types

Notice that 'veil-type' contains only one value, which is 'p' (partial), we will deal with this column in the data preprocessing section. There is another method that was used to explore the data, which to find number of unique values in each column and finding the most repeated value, a sample of columns is shown in the following figure:



	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color
count	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124
unique	2	6	4	10	2	9	2	2	2	12
top	e	x	y	n	f	n	f	c	b	b
freq	4208	3656	3244	2284	4748	3528	7914	6812	5612	1728

Figure 5: Dataset Description: No. of Rows and Unique Values, Top Repeated Value and How Much It Is Repeated

We can find valuable insights using the above figure, for example: 'gill-attachment' has 2 unique values, and the most repeated one is 'f' with frequency of 7914, which means that the other values frequency is 210 only!

Exploratory Visualization

Data Distribution

To know how the data is distributed, a count plot was used to indicate how many times deos a value in a column occure. This is the most suitible plot to start with as it is simple and easy to understand. It also gives us valuable information on how the values are distributed and the total count of each variable in each column.

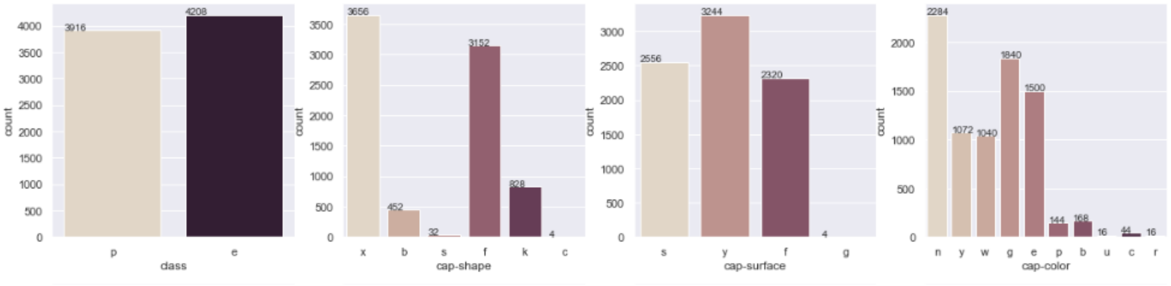


Figure 6: Sample of Data Distribution Plots

Plots are plotted here with column name at the button, X-axis holds column values, and Y-axis represents the total count of each value. From the above figure we can see that our classes are balanced, meaning that 'class' column values has close number of counts. The balance of classes are very important to avoid any bias during model building.

Take a look at the following plot:

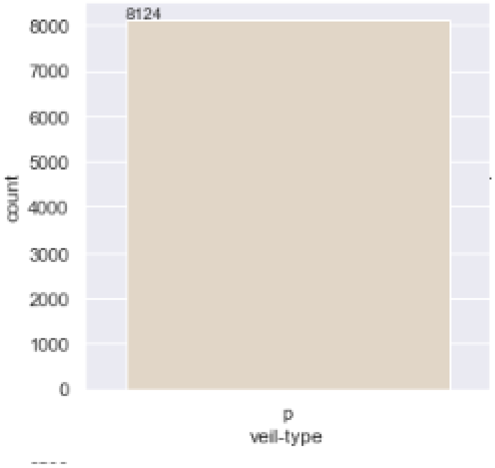


Figure 7: 'veil-type' Count Plot

As we mentioned earlier, the 'veil-type' column contains only one value, which is 'p' and this it is clearly shown using the count plot.

Plotting Each Feature Against The Target Variable ('class' Column):

Now, it is time to find (predict if any) relationships between each variable and the 'class' column, this is done by plotting each column twice: one against the 'e' (edible) value of 'class' and the other against 'p' (poisonous).

Here are the plots of 'odor' and 'spore-print-color':

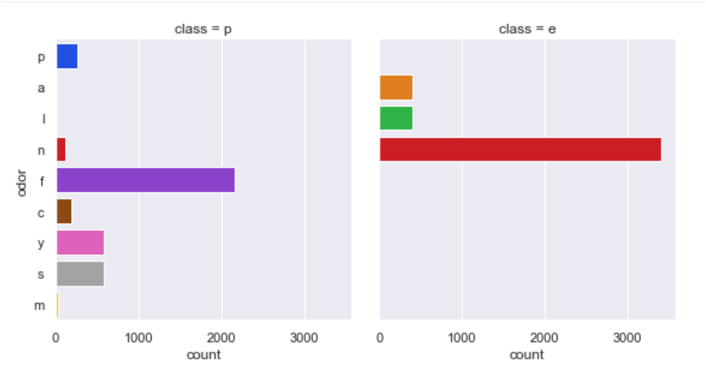


Figure 8: 'odor' Values Count in Each Class

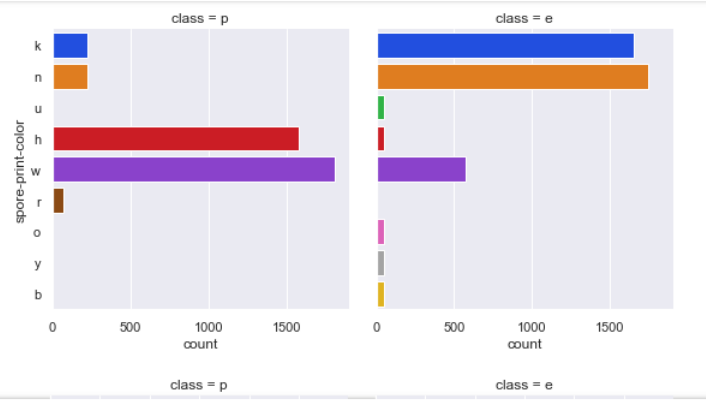


Figure 9: 'spore-print-color' Values Count in Each Class

These are some observation from this type of plots (including plots that are not in the above sample):

- 'bruises' has two values: t: true (has bruises) and f: false. The probability of a mushroom to be edible increases if the 'bruises' value is 't', it relates somehow to the mushroom class.
- 'odor' variable has a strong relationship with the target variable, because most of its values relates to one class. For example: (p, f, c, y, s, m) 'odor' values are only seen in the poisonous mushrooms! and (a, l) are seen only in edible ones. The only common value is n(none), which is obvious that in most of the time it belongs to the edible class.
- The 'b' (buff) value of the 'gill-color' is always classified as poisonouns.

- 'spore-print-color' will be an important varable in our model, as the 'odor' variable.

As 'spore-print-color' may be an important variable in our decision, it was nice to plot it with its values colors:

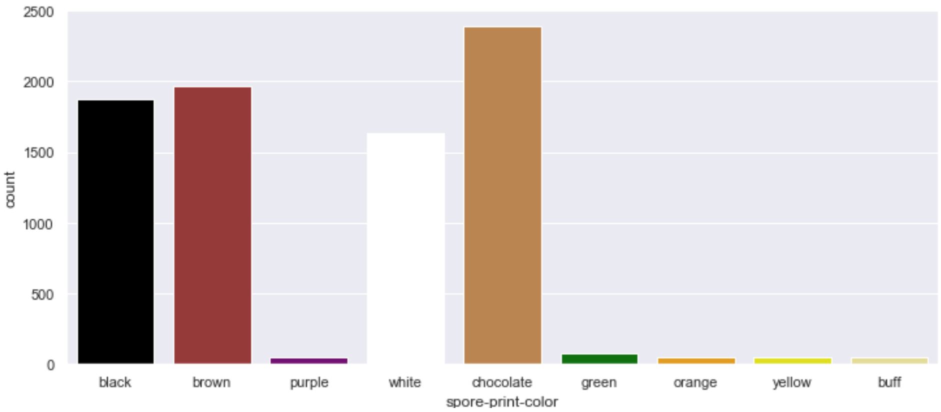


Figure 10: 'spore-print-color' Count Plot With Actual Colors

Algorithms and Techniques

Mushroom classification problem can be solved with supervised machine learning models, since all the data points are already labeled. The four algorithms that I used to solve this problem are: 'LogisticRegression', 'NaiveBayes', 'RandomForest' and 'Support Vector Machine'.

'LogisticRegression':

Logistic Regression is a spervised binary classification model that is widely used as a base model which makes it a good cndidate for our mushroom problem and that is how it works,

Logistic regression is a statistical technique used to predict probability of binary response based on one or more independent variables. It means that, given a certain factors (i.e our training set), logistic regression wil predict an outcome which has two values such as(? or ?) in our case, using mixumum likelihood estimation to calculate the regression co-efficients of the model and the model's features it will be ble to ome up with z, where $z = w0 + w1.x1 + w2.x2 + \ldots + wn.xn.$, where $w0, w1, w2, ..., wn$ are the regression co-efficients and $x1, x2, x3, ..., xn$ are our features.

After that the model calculates $F(z)$ which i sgiven by

$$F(z) = \frac{1}{1-e^{-z}}$$

$F(z)$ calculates the probability of the binary outcome into one of the two categories.

'NaiveBayes':

Naive Bayes uses probability theory and Bayes' Theorem which works on conditional probability. Conditional probability is the probability that something will happen, given that something else has already occurred. For example what is the probability that this mushroom is edible given that it's odor feature is pungent? Naive Bayes predicts membership probabilities for each class such as the probability that given record or data point belongs to a particular class. The class with the highest probability is considered as the most likely class.

'RandomForest':

which is an ensemble classifier that creates many decision trees and form randomly selected subset of training set. the nominated output class is the result of aggregated votes cieved by taking the mode of the classes output by individual trees because a single decision tree may be prone to a noise, but aggregate of many decision trees reduce the effect of noise giving more accurate results which is why I chosed this classifier along with it's ability to fight overfitting (compared to basic decision trees) by using bootstrap aggregation or bagging which reduces variance. The model was provided with 'random_state' value to maintain consistency.

'SVC':

Support Vector Machine (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, we will be using it for classification in our case. The idea of SVM is trying to find a hyperplane that best divides a dataset into two classes. we can think of a hyperplane as a line that linearly separates and classifiers a set of data. choosing the right hyperplane that can separate the two classes is based on the distance between the hyperplane and the nearest data point from either set which is known as the margin. The goal is to choose a hyperplane with the greatest possible margin between the hyperplane and any point within the training set, giving a greater chance of new data being classified correctly.

Most of the time data don't have a clear boundry separating it, this is where it can get trick in order to classify a dataset that is not linearly seperable. In simple terms the kerenel moves away from a 2d view of the data to a 3d view by lifting data points into higher dimention making it possible to hang a horizontal plane between the two sets. Support vector classifier was used for this problem because it is known to be robust and reliable.

Many machine learning algorithms cannot work with categorical data directly, so the categorical values were converted to numerical values by using one-hot-encoding method, this process converts categorical variables as binary vectors by creating a column for each possible value in dataset columns. I removed one column from each group while using this method to avoid dummy variable trap, which may negativly affect our results. K-Fold cross validation technique will be used to get more reliable scores. In K-fold cross validation, the original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k $-$ 1 subsamples are used as training data.[13]

Benchmark

The chosen benchmark model for this project is a simple “Logistic Regression” classifier. The mushroom dataset was introduced in Kaggle[4], and by examining different kernels I found that “Logistic Regression” is widely used to compare with. Also, the reason for choosing this specific algorithm is because it is simple and straightforward and doesn’t require any hyper-parameters to specify beforehand, and it is so popular for binary classification problems that almost every machine learning engineer has used before. Also, ’LogisticRegression’ is the most commonly reported data science method used at work for all industries except military and security where neural networks are used slightly more frequently according to “Kaggle: The State of ML and Data Science 2017”.[14]

III. Methodology

Data Preprocessing

This dataset doesn't need alot of preprocessing. However, these are the steps used to preprocess the data:

1. Removing 'veil-type' column becuase it doesn't affect the class of mushrooms (because it contains only one value).

2. Seperating the our target variable 'class' column into another variable to use it for splitting the dataset.

3. Dealing with missing values. The only column that contained missing values is 'stalk-root'. It has 2480 missing values, which are about 30.53% of the total points. Missing values here are represented using a question mark '?'. Since the ratio of missing values is large and deleting rows with missing values will delete one third of the data, I decided to keep the '?' as a value and train the model to see its results. Later, I found that results are very satisfying and there is no need to change the technique used for dealing with missing values.

4. Using One-Hot-Encoding (creating dummy variables) technique to convert categorical variables into binary vector by creating a column for each possible value in dataset columns. I removed one column from each group of encoded columns that correspond to one original column to avoid dummy variable trap, as mentioned in the 'Algorithms and Techniques' subsection.

5. Splitting the dataset into training and testing sets with testing ratio of 30%.

The final preprocessed dataset has 95 columns (after One-Hot-Encoding) and 8124 rows.

Implementation

As mentioned in the previous sections, the dataset was loaded and analysed to extract some useful information about it. Then, the dataset was cleaned and encoded as a part of making it suitible for all the model that we are going to use. The step that follows the datapreprocessing is splitting the dataset into two parts: 70% training and 30% testing. Now, it is time to implement our machine learning models, this is done using three main steps:

- **Fitting The Models:**

I started with creating a function that fits the given model using its training target and feature variables and records the training time of the model. The 'LogisticRegression' was the first model to be trained on the dataset. After that, all the models were trained one by one ('NaiveBaye', 'RandomForest', 'SVC'), and their training time was recorded.

- **Evaluating The Models Using Training and Testing Sets:**

  Here is the impotant part. I created another function for evaluating different models using different metrics according to their predictions. Each model is fed to this function which predicts the classes for training and testing sets, and the prediction time was recorded (I used training sets predictions to determine if a model was underfitting). Accuracy, recall, precision and F-measure performance metrics were recorded for both training and testing sets for each model.

- **Comparing The Results:**

  As mentioned in the metrics section, our main evaluation metrics will be the testing accuracy and false negatives count (considering poisonous as positive and edible as negative). After training the models and using them to predicts new points and recording their performance metrics a dataframe whose keys are models and values are these metrics, I displayed all the results in an easy and convenient way for the reader to determine the best performing model. I used a table, confusion matrices, and a plot of results to take the final decision about the best classifier.

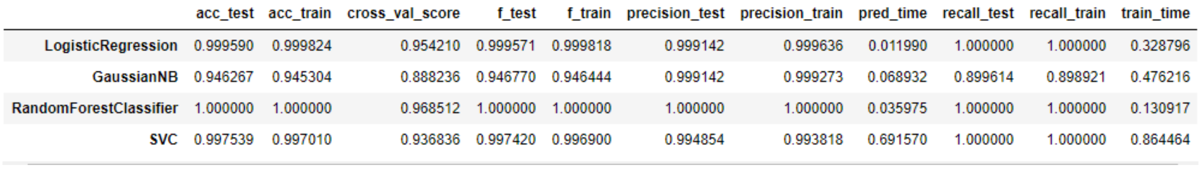
- **Enhaning The Model With The Highest Accuracy and Least False Negatives Count:**

  The chosen model is enhanced using hyper-parameter-tuning grid search technique.

Complications:

The implementation process ran smoothly without any problems, the only thing that worth mentioning that grid search took alot of time relativly.

Results:



	acc_test	acc_train	cross_val_score	f_test	f_train	precision_test	precision_train	pred_time	recall_test	recall_train	train_time
LogisticRegression	0.999590	0.999824	0.954210	0.999571	0.999818	0.999142	0.999636	0.011990	1.000000	1.000000	0.328796
GaussianNB	0.946267	0.945304	0.888236	0.946770	0.946444	0.999142	0.999273	0.068932	0.899614	0.898921	0.476216
RandomForestClassifier	1.000000	1.000000	0.968512	1.000000	1.000000	1.000000	1.000000	0.035975	1.000000	1.000000	0.130917
SVC	0.997539	0.997010	0.936836	0.997420	0.996900	0.994854	0.993818	0.691570	1.000000	1.000000	0.864464

Figure 11: Models Evaluation Metrics

The above table shows that 'RandomForestClassifier' is the best model. It has an accuracy of 100% at training and testing and 100% in all other metrices while using train_test_split, but when using cross validation it is accuracy is

96.85%. In terms of time, it was the fastest model to predict new data points. Interestingly, the 'LogisticRegression' model scored very well, in fact, except for 'RandomForestClassifier', it beated all other models with 99.98% training accuracy and 99.95% testing accuracy (in train_test_split), and with 95.42%, it was the second best model after 'RandomForestClassifier', and it is the least time consuming model in prediction.'GaussianNB' was the least scoring model with about 94.5% training accuracy and 94.6% testing accuracy (88.82% when using cross validation), and it has the shortest training time. 'SVC' has the longest training and predicting time with about 99.7% training and testing accuracy (93.68% when using cross validation). As the 'RandomForest' results are satisfying, the strategy of dealing with missing values will remain the same.

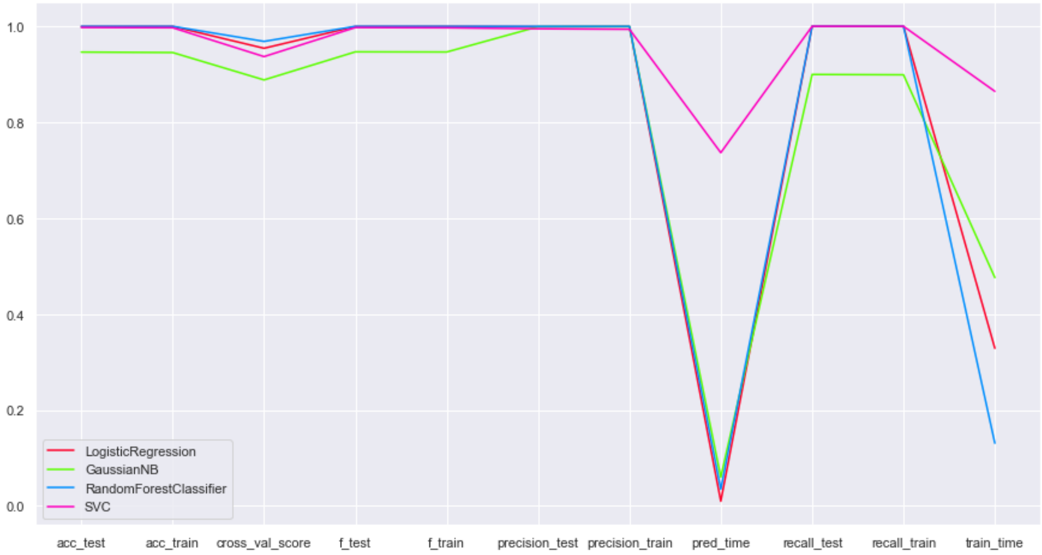


Figure 12: Models Evaluation Metrics Plot

Confusion Matrices:

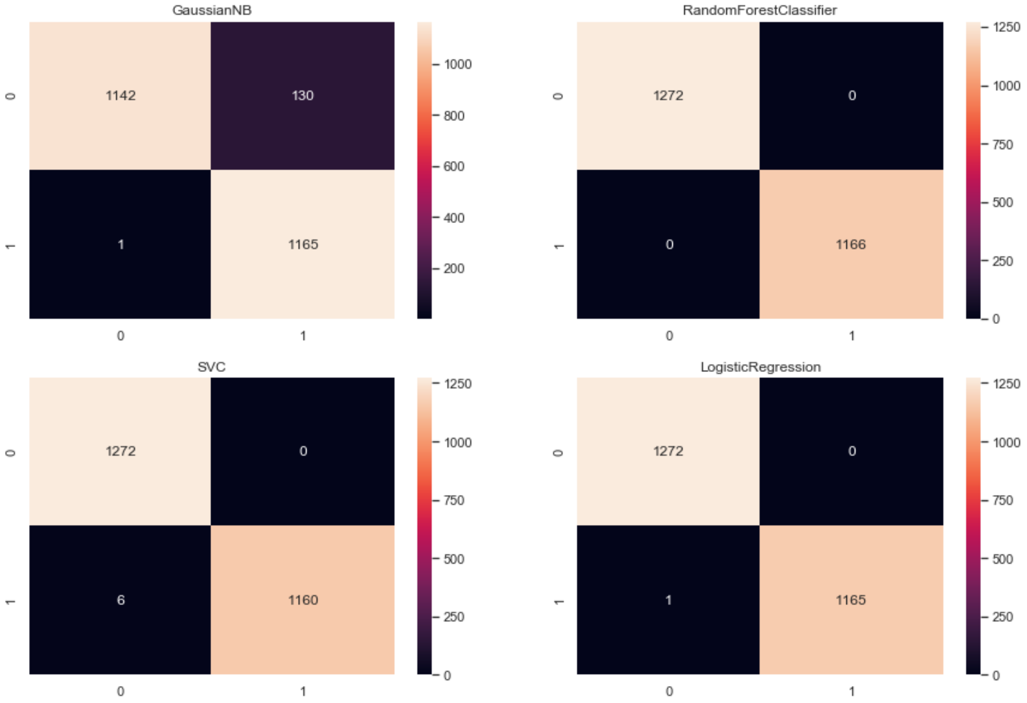


Figure 13: Confusion Matrices

False negatives:

LogisticRegression: 1, GaussianNB: 1, RandomForestClassifier: 0, SVC: 6.

The 'RandomForestClassifier' classified all the data points correctly and it has no false negatives. According to the above information, the 'RandomForestClassifier' was chosen as the best model.

Refinement

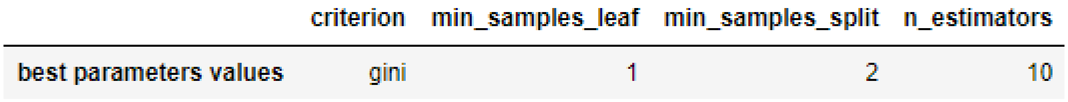
The 'RandomForestClassifier' needs refining to imporove its accuracy (because it was 96.85% when using cross validation). Hyper-parameter tuning was used to find out if there is better collection of parameters that may passed to the model and enhance the results, and to find the 'feature importances'.

I have tweaked the following parameters:

- 'n_estimators': the number of trees in the forest.
- 'criterion': the function to measure the quality of a split.

- 'min_samples_split': the minimum number of samples required to split an internal node.
- 'min_samples_leaf': the minimum number of samples required to be at a leaf node.

Random Forest Classifier Results:
Unoptimized Model: Testing Accuracy: 1.0000, cross Validation Score: 0.9685
Optimized Model: Testing Accuracy: 1.0000, cross Validation Score: 0.9685



	criterion	min_samples_leaf	min_samples_split	n_estimators
best parameters values	gini	1	2	10

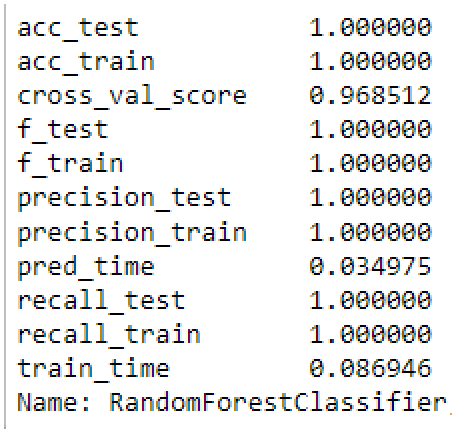
Figure 14: Best Parameters After Hyper-Parameter Tuning

The optimized model has the same accuracy as the default one. Also, the optimized model has the same parameters after tuning. Even the results are the same, we can use the optimized model to find important features in our problem.

IV. Results

Model Evaluation and Validation

The final model chosed for this problem is 'RandomForestClassifier' from sklearn library with its default parameters. I tried to hyper-parameter tune the model but the results was the same. The model has 0 false negatives (type II error).



```
acc_test            1.000000
acc_train           1.000000
cross_val_score     0.968512
f_test              1.000000
f_train             1.000000
precision_test      1.000000
precision_train     1.000000
pred_time           0.034975
recall_test         1.000000
recall_train        1.000000
train_time          0.086946
Name: RandomForestClassifier
```

Figure 15: Final Model Scores

The model can be considered robust because it is main performance rate (accuracy) was measured with two methods, train_test_split and cross validation. Using train_test_split method with 'RandomForest' output perfect scores with 100% accuracy, but this percentage cannot be trusted 100% because running the model with different random_states will some time change the percentage, and it depends heavly on how the data was splitted at first (especially when using random_state in train_test_split). I figured this problem by using K-fold cross validation technique, which ensures that all the data points will be treated as training or testing data point at some point of time, so by using this technique, we trained the model several times and averaged the scores to be able to trust the final results and to consider the model as robust after knowing its actual performance and output.

Justification

My Benchmark model is a defult logistic regression model that has a type II error of 1 and accuracy of 95.42%. The final model is an ensemble 'RandomForestClassifier' that has a type II error of 0 and accuracy of 96.85% which is higher than the benchmark model with about 1.43%. The final model has good performance rates, and as we mentioned before, the accuracy and false negatives count are our main focus. Because our model has beaten the benchmark model in both metrics with high scores, I think it will is a good model for mushroom classification problem.

V. Conclusion

Free-Form Visualization

Plotting Random Forest feature importance.

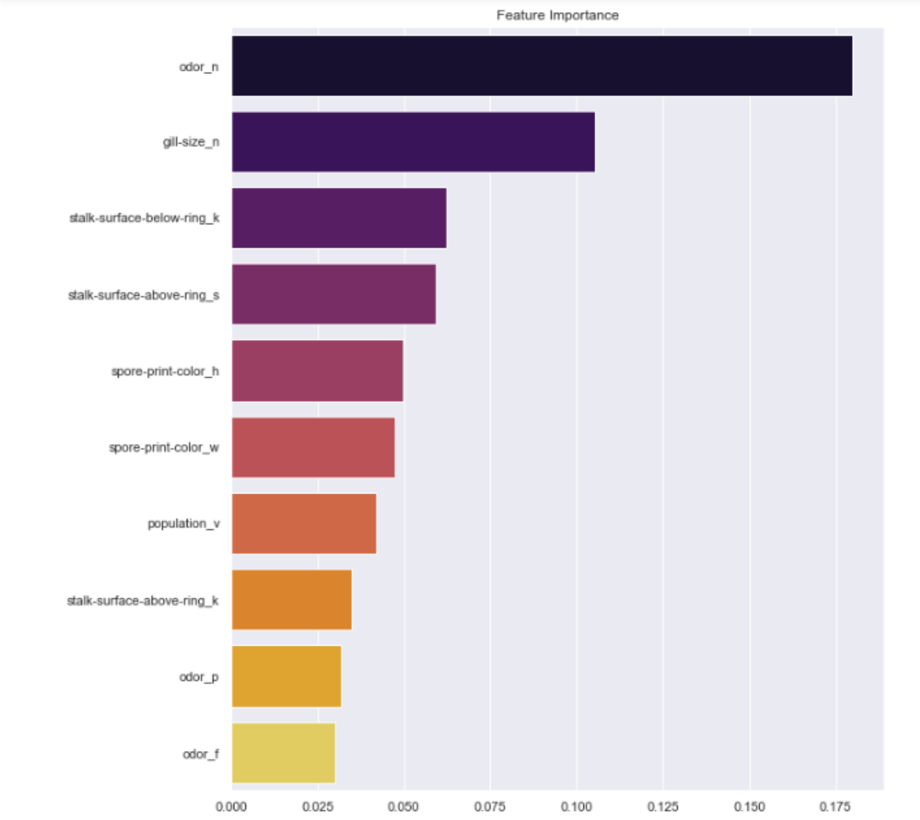


Figure 16: Feature Importance

Some values from 'odor', 'spore-print-color', 'stalk-surface-above-ring', 'gill-size', 'stalk-surface-below-ring' 'population', 'ring-type', 'ring-number' are considered the most 10 important factors in our classification. This is related to our observations about 'odor' and 'spore-print-color' found during examining the relationship between each column and the target variable in the 'Data Analysis' seciton. This is also a good indicator that our model is reliable.

Reflection

Main Project Steps:

1. Data Analysis: Reading the dataset file and knowing meta data.
2. Data Preprocessing: preparing dataset for machine learning models.
3. Splitting The Dataset: splitting dataset into training and testing.
4. Building The Benchmark Model: training 'LogistiecRegression' model.
5. Building Different Machine Learning Models: training 'NaiveBayes', 'RandomForest', 'SVC' models.
6. Evaluation: comparing different models results and choosing the best one among them.
7. Validation: enhancing the best model from the previous step.

Our problem was about classifying mushrooms as poisonous or edible. The dataset used for this problem was obtained from 'UCI Machine Leanring Repository'. It is a textual data with 23 columns and 8124 rows. All the columns contained categorical values, the 'class' column was our target variable. First: we started with data exploration to find the distribution of each column, and then we plotted every column against the target variable to see if we can predict the behavior of the model. Second: we preprocessed the data by using one-hot-encoding technique and removing unncessary columns. Third: We created the benchmark model (LogisticRegression), and built 3 other models (RandomForest, GaussianNaiveBayes, SVC) to make a comparison between all of them. LogisticRegression scored very well in terms of accuracy, but RandomForest was the best model becuase it didn't make any mistake during classification (100% training and testing accuracies and 0 false negatives). To confirm that, we also used cross validation.Finally, we implemented the hyper-parameter tuning technique to find the best parameters for our model, and to find the important features in our data.

It was difficult to me to visualize categorical-only variables to find valuable information or predict model's behavior or any important features that may affect the model performance. Finally, I found that count plot is the most suitible plot for our data.

Improvment

Improvements that will try do in future works:

- Doing feature reduction with PCA and possibly eliminating features that don't help in the classification process, and then comparing results with and without feature reduction.
- Using classification to predict and fill missing values, and then comparing the results of different machine learning model with and without missing values.
- Using more values in hyper-parameter tuning to find better models.
- Using more complex models like neural networks and compare its results with ours.

References

[1] *Mushroom - Wikipedia.* URL: https://en.wikipedia.org/wiki/Mushroom.

[2] *The health benefits of mushrooms | BBC Good Food.* URL: https://www.bbcgoodfood.com/howto/guide/health-benefits-mushrooms.

[3] *UCI Machine Learning Repository: Mushroom Data Set.* URL: https://archive.ics.uci.edu/ml/datasets/mushroom.

[4] *Mushroom Classification | Kaggle.* URL: https://www.kaggle.com/uciml/mushroom-classification/.

[5] *What is Logistic Regression? - Statistics Solutions.* URL: https://www.statisticssolutions.com/what-is-logistic-regression/.

[6] *6 Easy Steps to Learn Naive Bayes Algorithm (with code in Python).* URL: https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/.

[7] *Chapter 5: Random Forest Classifier - Machine Learning 101 - Medium.* URL: https://medium.com/machine-learning-101/chapter-5-random-forest-classifier-56dc7425c3e1.

[8] *Chapter 2 : SVM (Support Vector Machine) - Theory - Machine Learning 101 - Medium.* URL: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72.

[9] *Type I and type II errors - Wikipedia.* URL: https://en.wikipedia.org/wiki/Type_I_and_type_II_errors.

[10] *Metrics to Evaluate your Machine Learning Algorithm.* URL: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234.

[11] *Sensitivity and specificity - Wikipedia.* URL: https://en.wikipedia.org/wiki/Sensitivity_and_specificity.

[12] *F1 score - Wikipedia.* URL: https://en.wikipedia.org/wiki/F1_score.

[13] *Cross-validation (statistics) - Wikipedia.* URL: https://en.wikipedia.org/wiki/Cross-validation_(statistics)#k-fold_cross-validation.

[14] *The State of ML and Data Science 2017 | Kaggle.* URL: https://www.kaggle.com/surveys/2017.