# Machine Learning Engineer Nanodegree

## Capstone Project

Priyank Shah
June 6th, 2017

### Classification of Mushrooms as Edible or Poisonous.

## I. Definition

### Project Overview

The project is about applying different machine learning techniques to classify mushrooms in two distinct classes i.e. edible or poisonous. Classification of Mushrooms as poisonous or edible can be included in **healthcare** domain. Healthcare is currently in much need of Machine Learning techniques. There are lots of mundane things which are done manually and take precious time of healthcare personnel. These things can be automated by using machine learning techniques. For example it can be used to classify if a person is obese, underweight or normal by simply measuring certain features of the person and letting the software do the rest instead of manually applying the formulae and then calculating. Another simple example can be to classify various illness or diseases a person could be suffering from by giving the software relevant data such as blood report, full body checkup report etc. This in turn saves time of the doctor to manually go through each report to find the ailment. Furthermore the model could also accurately identify certain disease symptoms which might even be missed out by the doctor.

Coming to the classification of Mushroom, many of us like to go on camping or love to travel and eat new things. The most common thing which can be found in almost any forest is ‘mushroom’. At times if the traveler is stranded they can consume them for their survival. Here arises the problem that is that mushroom edible or not. Also not every person has absolute knowledge on mushroom. So for that simple problem you would not want to consult an expert every time or search on Google and find the same species (which can be quite tedious). And when you are in remote places there is high probability that the mushroom you selected can possibly be poisonous and may lead to fatality. As a machine learning model can work remotely without network as long as classification parameters are available, it is quite convenient to install an application that

does this. So next time when you are in a situation where you have to consume mushroom for survival, you can use the app and input its features and it will classify it as poisonous or edible. Thus training a simple machine learning model in this can help immensely in the classification. It can also be used when shopping for mushrooms through which you can be absolutely sure that it is edible. This is basically a classification problem and can be easily solved provided relevant and proper data.

As this problem is of binary classification simple methods can be used for classification. Methods include but not limited to Random Forest, Decision Tree Classifier, Support Vector Machines, Linear Regression, AdaBoost Classifier and Stochastic Gradient Descent. All of these methods of classification have been used and compared. The most appropriate model will then be selected based on precision of the models. Further analysis is done on the features of Dataset to determine which features are more important and have greater variance. This will be done with the help of Principle Component Analysis. Features will then be transformed into their Principle components. The transformed dataset are then be used to re-train the models. Few models will be selected based on the precision for optimization which is then be optimized by using Grid Search CV. Final selection is done for the models which have the highest precision and tested on testing set to get its performance.

Dataset used in this project was obtained from Kaggle ( https://www.kaggle.com ).Data set consists of 8129 samples of mushroom which are classified into either poisonous or edible.

## Problem Statement

Classification of Mushroom as edible or poisonous based on its physical traits. Mushroom are most common occurrences in remote and far off places in nature. People who like to travel and go to these places often find themselves stranded. For their survival they have to consume food available there. Mushroom is one such material. But the problem is has many species spread around the world and many of those are poisonous or unidentified. Thus by using machine learning techniques it can be correctly classified as poisonous or edible. Thus this problem is of binary classification. The classification is based on its physical traits such as colour, shape, etc. If the mushroom has certain specific traits it is classified either edible or poisonous. Many machine learning techniques can be used to classify them. The problem can be measured by normally used metrics such as precision, recall, f-beta, etc.

By applying Machine Learning models we can train the models in classifying mushroom, given its characteristics as input, as edible or poisonous. Thus, models are selected which are expected to perform well in classification problems, are simple and require less training time. After training the models will be evaluated based on certain metrics. Model with the best performance in metrics will be selected as the best model and can be applied to classification. Model selected should have very high performance in identifying poisonous mushrooms in particular.

## Metrics

Evaluation Metrics that will be used in this problem are,

1. **Precision** : Number of true positives (TP)(i.e. the number of items correctly labeled as belonging to the positive class) divided by the total number of elements labeled as belonging to the positive class (i.e. the sum of true positives(TP) and false positives(FP), which are items incorrectly labeled as belonging to the class). It can be calculated by the following expression **(TP/(TP+FP)).**
   It is normally used when classification of one class precedes over the other classes. In this model precision will have more importance as classification of poisonous mushrooms is very important

2. **Accuracy**: Number of correct predictions made as a ratio of all predictions made. It can be calculated by taking the ratio of correct predictions to the total number of predictions. The higher the accuracy the higher the reliability of the model in classification. But in skewed datasets it is not recommended. In this project model having high accuracy is preferred

3. **Recall:** Number of true positives(TP) divided by the total number of elements that actually belong to the positive class (i.e. the sum of true positives and false negatives(FN), which are items which were not labeled as belonging to the positive class but should have been).It can be calculated by the following expression **(TP/(TP+FN))**.It is used to measure the sensitivity of the model. As this project is more inclined towards classifying poisonous mushrooms correctly it does not have much importance in model evaluation. But once model can correctly classify poisonous mushrooms recall will be used as evaluation metric. Thus a model having higher value would be able to correctly identify all classes.

4. **F-1 score:** The F1 score (also F-score or F-measure) 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: p is the number of correct positive results divided by the number of all positive results, and r is the number of correct positive results divided by the number of positive results that should have been returned. The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst at 0.It can be calculated by using the following expression

   **f1 = 2*(p*r/(p+r))**.

   Thus during the final selection of the model, it is imperative for the model to have high F1 score. More high the score more better the resonance between precision and recall.

# II. Analysis

## Data Exploration

Dataset used in this project is taken from **Kaggle** (https://www.kaggle.com)

The link for the dataset is as follows:

https://www.kaggle.com/uciml/mushroom-classification

This dataset includes descriptions of hypothetical samples corresponding to **23 species** of gilled mushrooms in the **Agaricus** and **Lepiota** Family Mushroom drawn from The Audubon Society Field Guide to North American Mushrooms (1981). Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The dataset has features which are entirely **categorical** in nature. The dataset will be used after being transformed by **LabelEncoder**. The following describes the columns and its categorical values and what they represent.

**Attribute Information: (classes: edible=e, poisonous=p)**

**cap-shape**: bell=b, conical=c, convex=x ,flat=f, knobbed=k, sunken=s

**cap-surface**: fibrous=f, grooves=g, scaly=y, smooth=s

**cap-color**: brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y

**bruises**: bruises=t, no=f

**odor**:almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s

**gill-attachment**: attached=a, descending=d, free=f, notched=n

**gill-spacing**: close=c, crowded=w, distant=d

**gill-size**: broad=b, narrow=n

**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

**stalk-shape**: enlarging=e, tapering=t

**stalk-root**: bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?

**stalk-surface-above-ring**: fibrous=f, scaly=y, silky=k, smooth=s

**stalk-surface-below-ring**: fibrous=f, scaly=y, silky=k, smooth=s

**stalk-color-above-ring**: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y

**stalk-color-below-ring**: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y

**veil-type**: partial=p, universal=u

**veil-color**: brown=n, orange=o, white=w, yellow=y

**ring-number**: none=n ,one=o, two=t

**ring-type**: cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z

**spore-print-color**: black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y

**population**: abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y

**habitat**: grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d

The data set has **8124 samples**.

Number of samples with classification as ‘e’ i.e. **edible = 4208**.

Number of samples with classification as ‘p’ i.e. **poisonous = 3196**.

Thus it can be concluded that the data is not skewed and hence accuracy can be used as a valid metric. The data does not have any missing value

Below table shows how the data (few attributes and samples) is in the input file (csv):

| class | cap-shape | cap-surface | cap-color | bruises | odor | gill-attachment | gill-spacing | gill-size |
|---|---|---|---|---|---|---|---|---|
| P | x | s | n | t | p | f | c | n |
| E | x | s | y | t | a | f | c | b |
| E | b | s | w | t | l | f | c | b |
| P | x | y | w | t | p | f | c | n |
| E | x | s | g | f | n | f | w | b |

As you can observe the data is entirely formed of categorical values thus we first have to convert the data into numeric values corresponding to those categorical values. This is achieve by using Label encoder Following is the data after Label encoder is used:

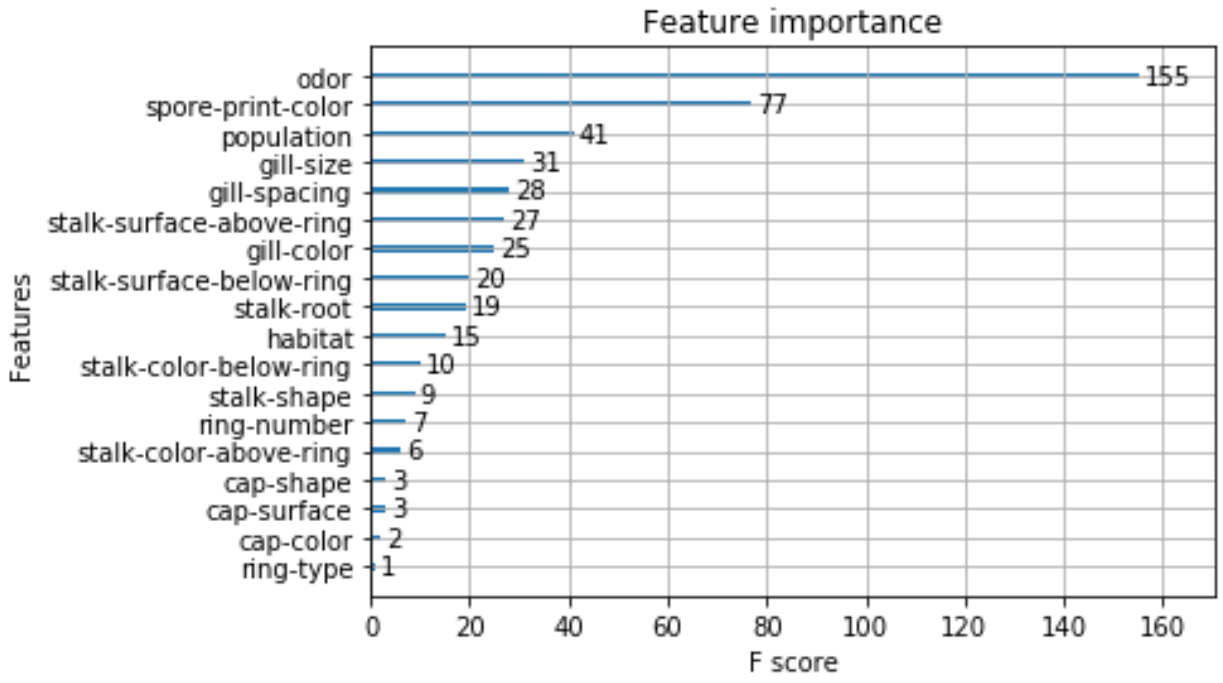
| class | cap-shape | cap-surface | cap-color | bruises | odor | gill-attachment | gill-spacing | gill-size |
|---|---|---|---|---|---|---|---|---|
| 1 | 5 | 2 | 4 | 1 | 6 | 1 | 0 | 1 |
| 0 | 5 | 2 | 9 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 2 | 8 | 1 | 3 | 1 | 0 | 0 |
| 1 | 5 | 3 | 8 | 1 | 6 | 1 | 0 | 1 |
| 0 | 5 | 2 | 3 | 0 | 5 | 1 | 1 | 0 |

Thus it can be seen that poisonous is given numeric value as ‘1’ and edible as ‘0’.

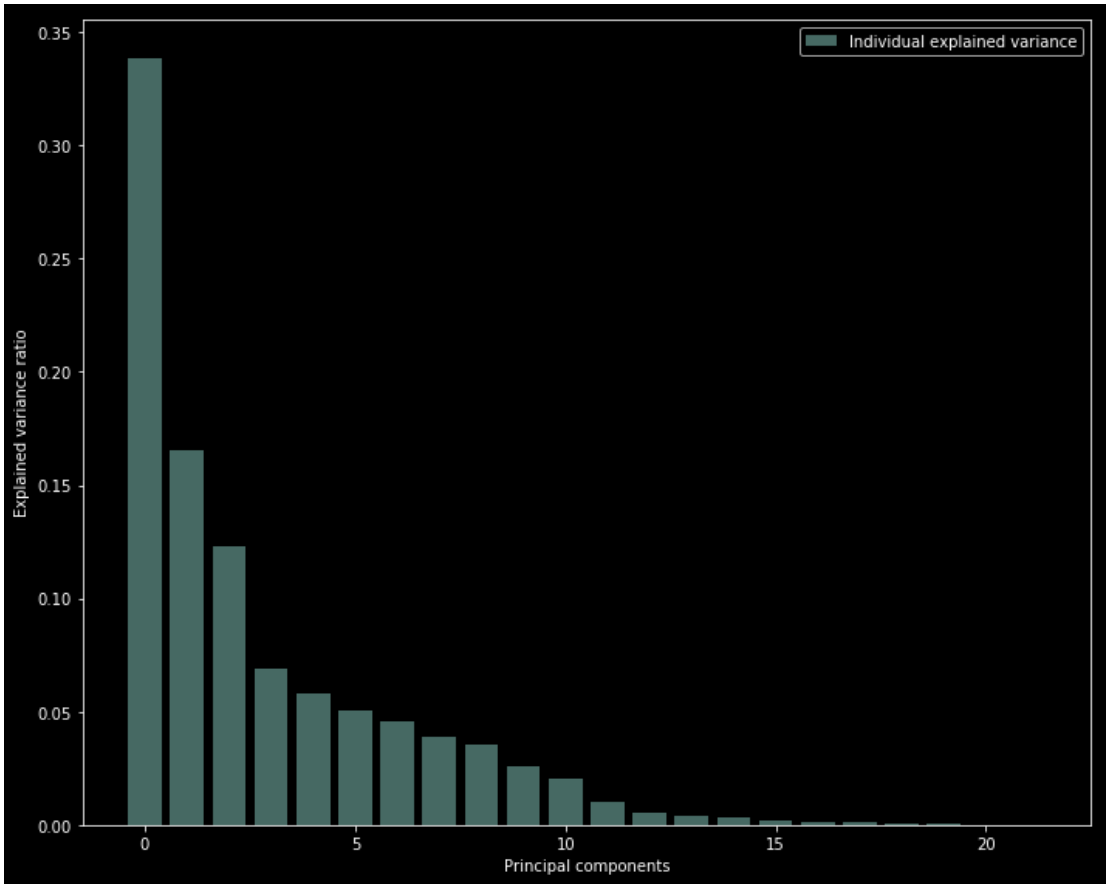
## Exploratory Visualization

As shown above data has categorical values corresponding to the type of mushroom and its physical trait.

The following plot shows the importance of each individual feature or characteristic of mushroom used in the dataset. This plot was obtained after training the dataset on XG Boost Classifier and afterwards using plot_importance function of XG Boost .




Thus it can be seen that ‘odor’ characteristics is the most important feature for classification and the least important feature could be said to be ‘ring-type’. Thus few bottom features contribute the least while classifying. Thus increasing training time and cost for the model. Hence there is a need to apply Principle Component Analysis to analyze which features contribute to variance of data. The following plot shows the data mapped to its principle components along with its Explained Variance Ratio. Higher the ratio higher is the contribution of the component in classification.




The plot shows that components 17 to 22 barely contribute anything to classification. Thus the data can be transformed into 16 principle components which cover about 99.5% of variance in data. Thus reducing dimensionality, training time and cost of the model. Since the dataset size used in this project is small there will not be any noticeable difference , but incase in future  dataset of size 100 times the present is used then it would make a much greater difference.

## Algorithms and Techniques

The problem is of Classification type so all the algorithms and techniques used are those that are normally or perform well in classification tasks. Furthermore problem is of Binary type classification so algorithms which give good precision and F1 score are used. All the algorithms used are taken from sci-kit learn library or can use sci-kit learn features.

Following algorithms or models have been used in this project:

1. **Logistic Regression**:

This model is mostly used for regression as well as classification type problems. Logistic Regression is an algorithm which makes distinct separation criterion between features. Logistic regression can be binomial, ordinal or multinomial. Binomial or binary logistic regression deals with situations in which the observed outcome for a dependent variable can have only two possible types, "0" and "1" (which may represent, for example, "dead" vs. "alive" or in our case “edible” vs. “poisonous”). Multinomial logistic regression deals with situations where the outcome can have three or more possible types (e.g., "disease A" vs. "disease B" vs. "disease C") that are not ordered. Ordinal logistic regression deals with dependent variables that are ordered. In binary logistic regression, the outcome is usually coded as "0" or "1", as this leads to the most straightforward interpretation. If a particular observed outcome for the dependent variable is the noteworthy possible outcome (referred to as a "success" or a "case") it is usually coded as "1" and the contrary outcome (referred to as a "failure" or a "noncase") as "0". Logistic regression is used to predict the odds of being a case based on the values of the independent variables (predictors). The odds are defined as the probability that a particular outcome is a case divided by the probability that it is a noncase.

Like other forms of regression analysis, logistic regression makes use of one or more predictor variables that may be either continuous or categorical. Unlike ordinary linear regression, however, logistic regression is used for predicting binary dependent variables rather than a continuous outcome. Given this difference, the assumptions of linear regression are violated. In particular, the residuals cannot be normally distributed. In addition, linear regression may make nonsensical predictions for a binary dependent variable. What is needed is a way to convert a binary variable into a continuous one that can take on any real value (negative or positive). To do that logistic regression first takes the odds of the event happening for different levels of each independent variable, then takes the ratio of those odds (which is continuous but cannot be negative) and then takes the logarithm of that ratio. This is referred to as logit or log-odds) to create a continuous criterion as a transformed version of the dependent variable. Thus the logit transformation is referred to as the link function in logistic regression—although the dependent variable in logistic regression is binomial, the logit is the continuous criterion upon which linear regression is conducted.

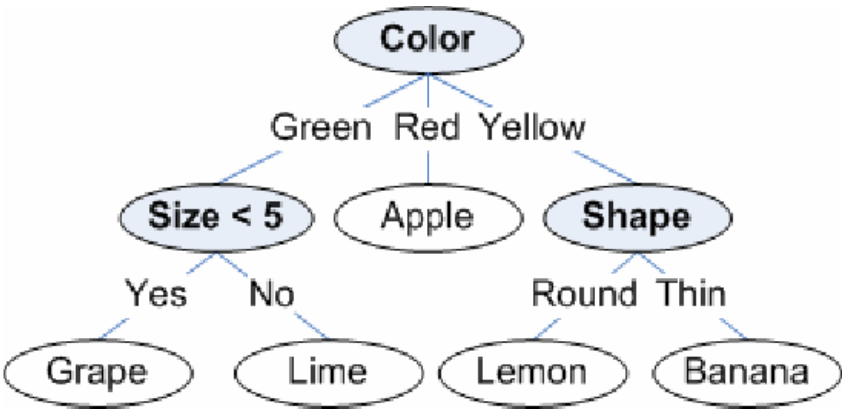
The logit of success is then fitted to the predictors using linear regression analysis. The predicted value of the logit is converted back into predicted odds via the inverse of the natural logarithm, namely the exponential function. Thus, although the observed dependent variable in logistic regression is a zero-or-one variable, the

logistic regression estimates the odds, as a continuous variable, that the dependent variable is a success (a case). In some applications the odds are all that is needed. In others, a specific yes-or-no prediction is needed for whether the dependent variable is or is not a case; this categorical prediction can be based on the computed odds of a success, with predicted odds above some chosen cutoff value being translated into a prediction of a success.

2. **Decision Tree Classifier**:

Decision tree learning uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It is one of the predictive modeling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a finite set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. This algorithm is the most common and widely used algorithm for classification type problems which have categorical values.

A decision tree or a classification tree is a tree in which each internal (non-leaf) node is labeled with an input feature. The arcs coming from a node labeled with an input feature are labeled with each of the possible values of the target or output feature or the arc leads to a subordinate decision node on a different input feature. Each leaf of the tree is labeled with a class or a probability distribution over the classes.A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. For example see a simple Decision Tree shown below:



Algorithms for constructing decision trees usually work top-down, by choosing a variable at each step that best splits the set of items. Different algorithms use different metrics for measuring "best". These generally measure the homogeneity of the target variable within the subsets. Technique used for determining separation in features is called 'Gini Impurity'.

It is used by the CART (classification and regression tree) and Decision stream algorithms, Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. In this project Decision Tree Classifier is used with 'gini' impurity as its default parameter.

3. **Random Forest Classifier**:

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e. have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance.This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, \ldots, x_n$ with responses $Y = y_1, \ldots, y_n$, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For b = 1, ..., B:

1. Sample, with replacement, B training examples from X, Y; call these $X_b$, $Y_b$.
2. Train a decision or regression tree $f_b$ on $X_b$, $Y_b$.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

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

or by taking the majority vote in the case of decision trees.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, if the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

The number of samples/trees, B, is a free parameter. Typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set. An optimal number of trees B can be found using cross-validation, or by observing the out-of-bag error: the mean prediction error on each training sample $x_i$, using only the trees that did not have $x_i$ in their bootstrap sample. The training and test error tend to level off after some number of trees have been fit.

The above procedure describes the original bagging algorithm for trees. Random forests differ in only one way from this general scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated. Typically, for a classification problem with p features, √p (rounded down) features are used in each split.

4. **AdaBoost Classifier**:

   AdaBoost, short for "Adaptive Boosting", is a machine learning meta-algorithm. It can be used in conjunction with many other types of learning algorithms to improve their performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In some problems, it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can

be weak, but as long as the performance of each one is slightly better than random guessing (e.g., their error rate is smaller than 0.5 for binary classification), the final model can be proven to converge to a strong learner.

While every learning algorithm will tend to suit some problem types better than others, and will typically have many different parameters and configurations to be adjusted before achieving optimal performance on a dataset, AdaBoost (with decision trees as the weak learners) is often referred to as the best out-of-the-box classifier. When used with decision tree learning, information gathered at each stage of the AdaBoost algorithm about the relative 'hardness' of each training sample is fed into the tree growing algorithm such that later trees tend to focus on harder-to-classify examples.

AdaBoost refers to a particular method of training a boosted classifier. A boost classifier is a classifier in the form

$$F_T(x) = \sum_{t=1}^{T} f_t(x)$$

where each $f_t$ is a weak learner that takes an object x as input and returns a value indicating the class of the object. For example in the two class problem, the sign of the weak learner output identifies the predicted object class and the absolute value gives the confidence in that classification. Similarly, the *T*th classifier will be positive if the sample is believed to be in the positive class and negative otherwise.

Each weak learner produces an output, hypothesis $h(x_i)$ for each sample in the training set. At each iteration *t* , a weak learner is selected and assigned a coefficient $\alpha_t$ such that the sum training error Et of the resulting t-stage boost classifier is minimized.

$$E_t = \sum_i E[F_{t-1}(x_i) + \alpha_t h(x_i)]$$

Here $F_{t-1}(x)$ is the boosted classifier that has been built up to the previous stage of training, E(F) is some error function and $f_t(x) = \alpha_t h(x)$ is the weak learner that is being considered for addition to the final classifier.

5. **XG Boost Classifier**:

XGBoost is an algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

The XGBoost library implements the gradient boosting decision tree algorithm. This algorithm goes by lots of different names such as gradient boosting, multiple additive regression trees, stochastic gradient boosting or gradient boosting machines. Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made.

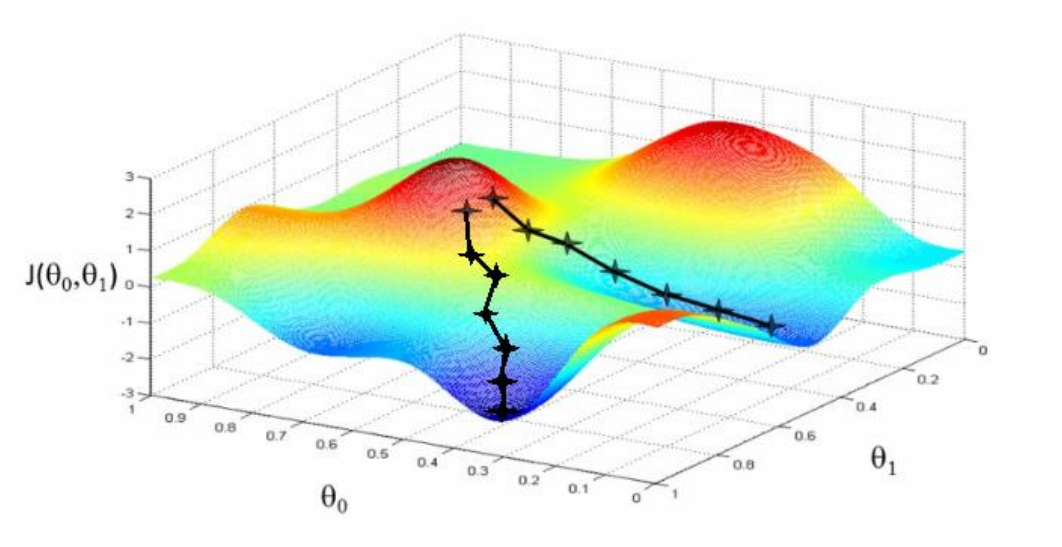
A popular example is the AdaBoost algorithm that weights data points that are hard to predict. Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.This approach supports both regression and classification predictive modeling problems.

6. **Stochastic Gradient Descent**:

This algorithm is quite similar to Gradient Descent algorithm where the algorithm tries to minimize error function by making small changes to weights. This algorithm is different in the sense that it takes small batches of dataset and tries to fit to the dataset, although it takes a lot more iterations for it to reach minima , it gives surprisingly good and accurate result. Choosing of batches is done randomly with the size of the batches remaining the same.

The algorithm initializes a set of weights which are then fitted to the dataset and error is calculated as mismatch in the classification. Based on the error and the learning rate assigned it changes the weights which are then again fitted to the data set and the error is found. This process repeats itself until it has found lowest error or the error does not decrease or the number of iteration gets completed.

Thus it follows Gradient Descent way of approach by gradually decreasing the error each step. Following figure shows gradient descent :




The above figure shows two features $\theta_0$ and $\theta_1$ and the error function $J(\theta_0,\theta_1)$ plotted. It can be seen that the algorithm takes small steps from the peaks as it gradually descends and reaches to the lower valley (minima). One major disadvantage that it faces is of local minima. But in stochastic gradient descent it gets erased by using batches so if one batch gets stuck in local optimum rest of the batches have high probability to reach the actual minima.

Weights are changed based on the following formula:

$$W = W - \eta \Delta W$$

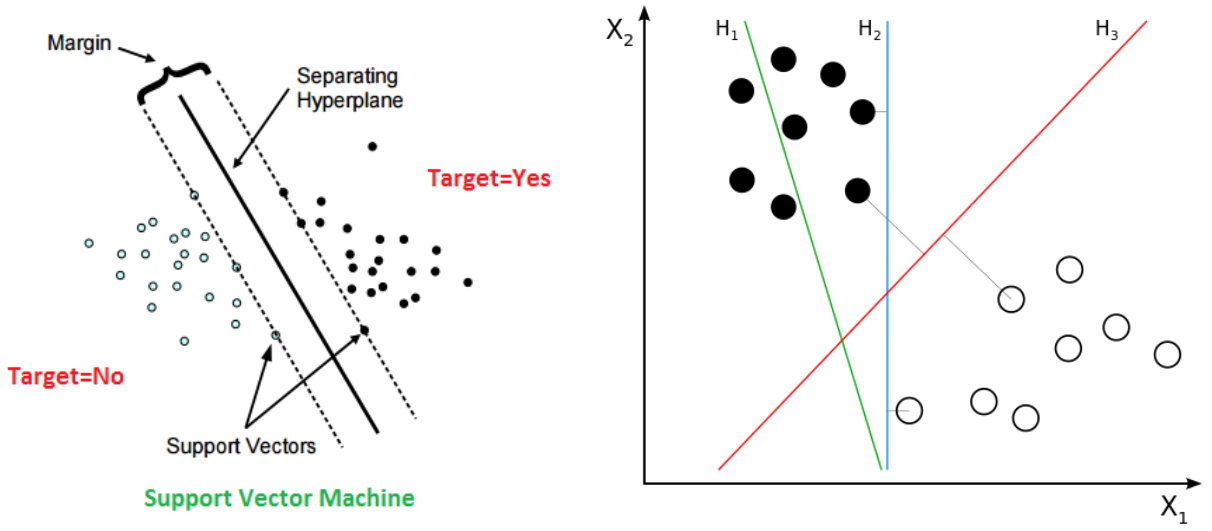
Where $\Delta W$ = The change in the weights as per error function, $\eta$ = Learning rate which determines how fast the algorithm should adapt to the change.

7. **Support Vector Machines**:

Support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are

then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. Following figure shows how SVM is different from Linear classifier.




The above figure shows a SVM classifier it does not just classify the cases like linear classifier, it classifies it while having an appropriate margin as shown in the figure. So, the classifier is much more robust than linear classifier. Both H2 and H3 classify the models correctly but looking at the plot it can be concluded that H3 is the better fir. This is achieved due to SVM having Margin between both the classes.

More formally, a support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

Whereas the original problem may be stated in a finite dimensional space, it often happens that the sets to discriminate are not linearly separable in that space. For this reason, it was proposed that the original finite-dimensional space be mapped into a much higher-dimensional space, presumably making the separation easier in that space. To keep the computational load reasonable, the mappings used by SVM schemes are designed to ensure that dot products may be computed easily in terms

of the variables in the original space, by defining them in terms of a kernel function k(x,y) selected to suit the problem.

Following Techniques are used for Analysis and parameter optimization for the models:

1. **Train Test Split**:

   This is a function provided by sklearn for the sole purpose of splitting the data into training and testing sets. This is done to ensure that model is evaluated on testing set and not on the training set. Since the model is trained on the training set it fits the training set. And when the model faces real time data it performs poorly. Thus, evaluating the model on testing data which it has not seen is the best method for selecting models. Split is done randomly.

2. **Principle Component Analysis (PCA)**:

   Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components (or sometimes, principal modes of variation). The number of principal components is less than or equal to the smaller of the number of original variables or the number of observations. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables. There is a function called PCA, provided by sklearn which has been used in the project to determine the principle compenet variance. And selecting those components which contribute to the variance. Thereby reducing the dimensionality of the model.

3. **Grid Search CV**:

   It is a technique made available by sklearn wherein you give model and sets of parameters to be used in the model for tuning the model. Parameters are provided in dictionary format in this project. The parameters are used by the function in the model to find out which parameters give the highest score.

Scoring function can also be provided to GridSearchCV. In this project scoring function used is Precision.

### Benchmark

The benchmark model in this case would be to classify every mushroom as poisonous. Even though it is not a valid method for classification but it still gives 50% accuracy in prediction as it is a binary classification. This gives a precision of 0.5, recall of 0.5, f-1 score of 0.5 all of which is quite less. Ideally the model should interpret the features of mushroom and then classify weather the mushroom is edible or not. The model can be evaluated based on the f-1 score as it is the harmonium of precision and recall. But more importantly precision should be considered as higher the precision the better the classification of one class, which is of poisonous mushrooms.

## III. Methodology

### Data Preprocessing

Dataset is imported from **CSV** file using read_csv function of pandas. Thus the dataset obtained is in dataframe format. Dataset used in this project does not have any NULL values nor is it missing any values. But the data is entirely categorical. Thus there is a need to convert the data into numeric values.

This is obtained by using **LabelEncoder** feature of sklearn. It gives a unique value to each distinct variable in a column. So the data is then transformed into numeric values a column at a time. The dataset is then split into labels which includes 'class' attribute of the data set and samples which consists of the rest of the attributes. Now that the data is split into training samples and labels it can be directly fed to the training algorithms which.

Encoded Dataset sample is shown below:

| class | cap-shape | cap-surface | cap-color | bruises | odor | gill-attachment | gill-spacing | gill-size |
|---|---|---|---|---|---|---|---|---|
| 1 | 5 | 2 | 4 | 1 | 6 | 1 | 0 | 1 |
| 0 | 5 | 2 | 9 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 2 | 8 | 1 | 3 | 1 | 0 | 0 |
| 1 | 5 | 3 | 8 | 1 | 6 | 1 | 0 | 1 |
| 0 | 5 | 2 | 3 | 0 | 5 | 1 | 1 | 0 |

## Implementation

Implementation of the project is done as per the following steps:

1. **Getting Data**:

Data has been obtained from Kaggle. The dataset was in CSV (comma separated values) which was then downloaded and kept in the same path as that of Anaconda notebook.

2. **Getting to Know Data**:

Gathered dataset is then analysed. Analysis is done by using various functions of Pandas library. Data is imported to the notebook by using 'read_csv()' function with default parameters. Dataset imported is a DataFrame object of pandas. First Few rows of data are viewed by using 'head()' function of DataFrame object. Then each column is checked for counts ,missing values using 'describe()' function. Also each column is checked for the distinct values occurring and its count by using 'value_counts()' function. The data set contained 23 columns and 8129 samples. After the analysis it is found that the data is entirely made up of categorical values. So it was concluded that categorical values have to be converted into numeric values so that all the models can be used.

3. **Split Dataset into Samples and Labels**:

Dataset is then split into Labels (containing classification result as edible or poisonous) and samples (containing rest of the attributes or features ) amounting to 22 columns.

4. **Encoding the data into numerical values**:

Split dataset is then encoded into numerical values. Each numeric value corresponding to distinct categorical values in each column. This was achieved by using LabelEncoder function from sklearn.preprocessing. Encoding was done one column at a time.

5. **Training, Validation, Testing Split in Dataset**:

Dataset is then split into training, validation and testing set. This is done so that a model does not overfit to the training data. Validation data split is done so as to select the type of models which perform good in validation set after having trained in training set. But still there is one more problem which arises that is bleeding of the validation data into the training data. This happens when one tries to maximize the performance in validation set by tweaking the parameters in training set. Thus, the model tries to indirectly fit the validation set. Hence, the need for a set which can be used for testing and which is isolated from training is required. Performance on this set will be the final deciding factor for selection of the best model.

Split is done by using a function provided by sklearn.model_selection called 'train_test_split' which randomly splits the data into 2 different sets. Thus dataset was first split into 70% and 30% sets, 70% set being training. 30% set was further split into 15% set each for validation and testing sets. Following are the sample counts obtained for the data split:

Training data count: 5686

Validation data count: 1219

Testing data count: 1219

6. **Getting Performance of Benchmark Model**:

As previously described that benchmark model to be used in this project is to make all predictions as '1' i.e. poisonous. Performance metrics such as Precision, F1-score, Accuracy and Recall were used to get performance of benchmark model. Metrics are calculated by using their specific functions from sklearn.metrics. Following are the values obtained:

**Precision**: 0.490566

**Accuracy**: 0.490566

**Recall**: 1.000000

**F1 score**: 0.658228

7. **Importing and Initializing different models for learning**:

Following models are imported from different libraries and are initialized to their default parameters:

1. **Logistic Regression**:

**Importing Library**: sklearn.linear_model

**Default parameters**: LogisticRegression (C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1, penalty='l2', random_state=None, solver='liblinear', tol=0.0001, verbose=0, warm_start=False)

2. **Decision Tree Classifier**:

**Importing Library**: sklearn.tree

**Default parameters**: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')

3. **Random Forest**:

**Importing Library**: sklearn.ensemble

**Default parameters**: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)

4. **Support Vector Machines**:

**Importing Library**: sklearn.svm

**Default parameters**: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma='auto', kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)

5. **AdaBoost Classifier**:

**Importing Library**: sklearn.ensemble

**Default parameters**: AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0, n_estimators=50, random_state=None)

6. **XG Boost Classifier**:

**Importing Library**: xgboost

**Default parameters**: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3, min_child_weight=1, missing=None, estimators=100, n_jobs=1, nthread=1, objective='binary:logistic', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=0, silent=True, subsample=1)

7. **Stochastic Gradient Descent**:

**Importing Library**: sklearn.linear_model

**Default parameters**: SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1, eta0=0.0, fit_intercept=True, l1_ratio=0.15, learning_rate='optimal', loss='hinge', n_iter=5, n_jobs=1, penalty='l2', power_t=0.5, random_state=None, shuffle=True, verbose=0, warm_start=False)

7. **Training Dataset on the initialized model**:

All of the above models have been fit to the training data by using 'fit()' function in sklearn. This fits the data to the model and can be evaluated using inbuilt score function or using custom metric functions such as precision, f1-score, accuracy, recall.

8. **Obtaining Performance metrics of the models**:

Performance metrics evaluated are imported through sklearn.metrics library. These metrics are Precision, F1-score, accuracy, Recall. Metrics are calculated by providing predicted labels for validation samples and original labels. Prediction is obtained by using predict() function of the model.

9. **Principle Component Analysis (PCA)**:

Using sklearn's PCA technique from sklearn.decomposition to find principle components and its corresponding individual explained variance. Then analyzing the components and performing dimension reduction by selecting certain components which cover 99.5% of the data variance and performing fit_transform() function of PCA.

10. **Training the models on PCA transformed data and evaluating performance.**

11. **Selecting models which have high performance.**

12. **Performing parameter optimization on selected models**:

Performance optimization is achieved by using Grid Search CV technique from sklearn.model_selection library. Grid Search CV is provided the model, parameters to be optimized and the scoring function which is used to select best parameters from the provided parameters.

13. **Training and evaluating performance based on the optimal parameter obtained**:

This is done for both the validation and test sets

14. **Selecting the best model/models based on the performance achieved in the test set.**

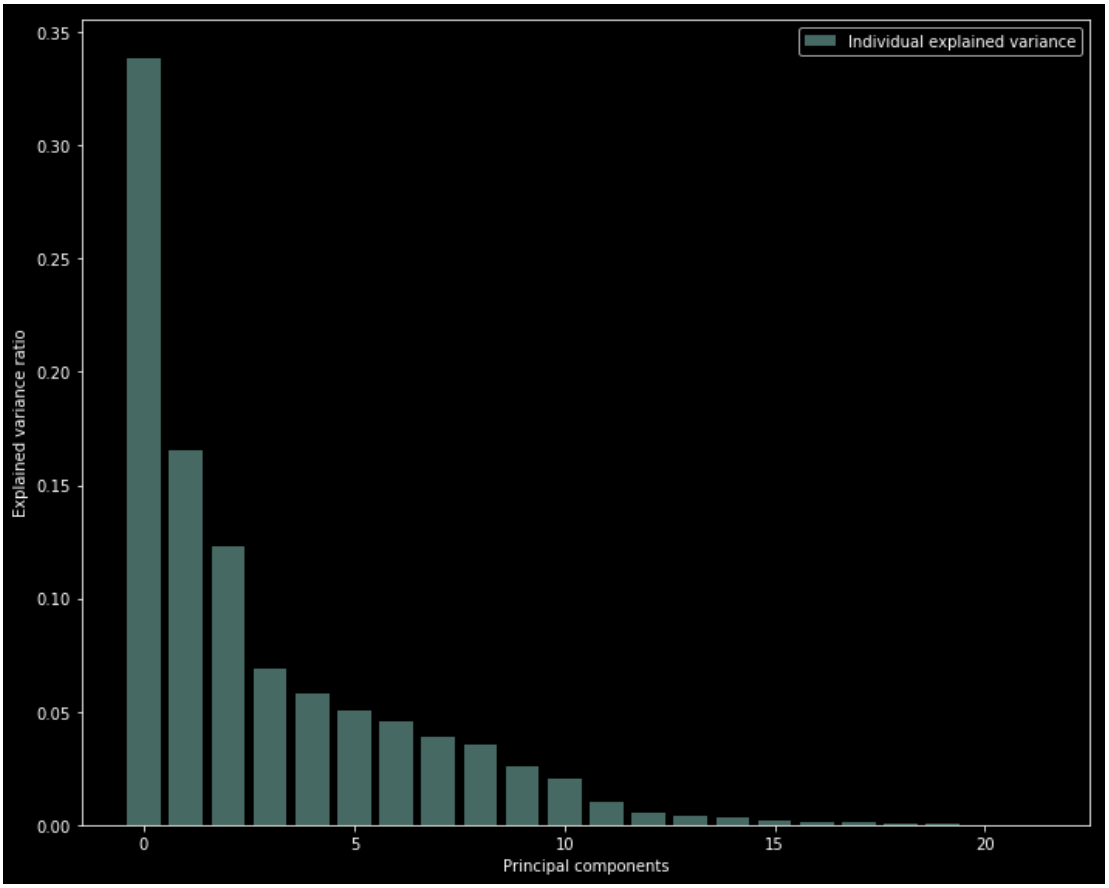
## Refinement

There are two major types of refinement done in this project.

1. Dimensionality Reduction
2. Parameter optimization

1. **Dimensionality Reduction**:

   This is achieved by using PCA technique. Initially PCA is performed with components as 22(without reduction). Following is the plot obtained of Explained Variance Ratio.




The above plot shows each principle component with its variance ratio. Components 17-22 do not contribute much to the overall variance of the data. Just 1-16 components alone contribute to about 99.5% of the data variance. Thus dataset is transformed into 16 principle components by initializing 'n_components' variable in PCA as 16. The transformed data has thus undergone dimensionality reduction and will result in ease of training and fitting models on a less dimension dataset.

2. **Parameter Optimization**:

This is achieved by using Grid Search CV technique found in sklearn.model_selection. A set of parameters are provided from which optimum parameters have to be selected along with the training model and the scoring function. Grid Search CV has a function called best_params_ which returns best parameters from the set of parameters provided. Grid Search CV was performed for: Support Vector Machines, Random Forest, AdaBoost Classifier, XG Boost Classifier and Decision Tree Classifier.

As the models could already almost perfectly classify the data there was not much increase in their performance by using Grid Search CV.

Following are the set of parameters used:

Parameters for Support Vector Machine

```
parameters_svr =
{'C':[0.1,0.3,0.9,1.0,3.0,9.0,10.0],'kernel':('linear','rbf'),'random_state':[1,42,56,32,15]}
```

Parameters for Random_forest

```
parameters_ran = {'n_estimators':[2,5,7,10,12,15,18,20],'random_state':[1,42,56,32,15]}
```

Parameters for XG Boost Classifier

```
parameters_xgb =
{'learning_rate':[0.01,0.03,0.09,0.1,0.3,0.9,1],'booster':('gbtree','gblinear','dart'),
'random_state':[1,15,2,3,48,42]}
```

Parameters for Decision Tree Classifier

```
parameters_dt = {'min_samples_split':[2,7,10], 'random_state':[1,42,56,32,15]}
```

Parameters for AdaBoost Classifier

```
parameters_ada={'learning_rate':[0.01,0.03,0.09,0.1,0.3,0.9,1],'n_estimators':[10,30,50,7
0],'random_state':[1,42,56,32,15]}
```

# IV. Results

## Model Evaluation and Validation

1. **Following table shows the performance of all the models with default parameters**:

| Models | Accuracy | F1-score | Precision | Recall |
|---|---|---|---|---|
| XGBoost Classifiier | 1.00000 | 1.00000 | 1.00000 | 1.00000 |
| Decision Tree Classifier | 1.00000 | 1.00000 | 1.00000 | 1.00000 |
| Stochastic Gradient Descent | 0.93683 | 0.93535 | 0.91612 | 0.95540 |
| AdaBoost Classifier | 1.00000 | 1.00000 | 1.00000 | 1.00000 |
| Logistic Regression | 0.93847 | 0.93584 | 0.93345 | 0.93825 |
| Random Forest | 1.00000 | 1.00000 | 1.00000 | 1.00000 |
| Support Vector Machines | 1.00000 | 1.00000 | 1.00000 | 1.00000 |

For this performance analysis, training was done on training sets and prediction on validation set. The above performance evaluation for all the models shows that all the models selected except for Logistic Regression and Stochastic Gradient Descent are suitable when applied default parameters. Also majority of the models can successfully classify all the examples correctly since F1 score is 1.0. Thus models selected comfortably surpass benchmark model.

2. **Following table shows the performances of the models when the training set has been transformed into 16 principle components with the help of PCA(with default parameters)**:

| Models | Accuracy | F1-score | Precision | Recall |
|---|---|---|---|---|
| XGBoost Classifiier | 0.99180 | 0.99136 | 0.99826 | 0.98456 |

| | | | | |
|---|---|---|---|---|
| Decision Tree Classifier | 0.98934 | 0.98886 | 0.98801 | 0.98971 |
| Stochastic Gradient Descent | 0.87039 | 0.86855 | 0.84330 | 0.89537 |
| AdaBoost Classifier | 0.96637 | 0.96493 | 0.96246 | 0.96741 |
| Logistic Regression | 0.88023 | 0.87326 | 0.88401 | 0.86278 |
| Random Forest | 0.99672 | 0.99656 | 1.00000 | 0.99314 |
| Support Vector Machines | 1.00000 | 1.00000 | 1.00000 | 1.00000 |

For this performance analysis, training was done on training sets and prediction on validation set. After dimensionality reduction (22 to 16 components) it can be said that performances of all the models except for SVM decreases. And Stochastic Gradient Descent and Logistic Regression models precision goes below 0.90 which is very low for critical classification of mushrooms. Thus these two models will not be analysed further.

3. **Following table shows performances of models with 16 principle components with parameter optimized**:

| Models | Accuracy | F1-score | Precision | Recall |
|---|---|---|---|---|
| XG Boost Classifier | 1.00000 | 1.00000 | 1.00000 | 1.00000 |
| Support Vector Machines | 1.00000 | 1.00000 | 1.00000 | 1.00000 |
| Random Forest | 1.00000 | 1.00000 | 1.00000 | 1.00000 |
| AdaBoost Classifier | 0.97785 | 0.97674 | 0.98097 | 0.97256 |

| | | | | |
|---|---|---|---|---|
| Decision Tree Classifier | 0.98934 | 0.98886 | 0.98801 | 0.98971 |

For this performance analysis, training was done on training sets and prediction on validation set. It can be compared that average performances of all the models have gone up after optimization. Thus it could be estimated that one of these three models i.e. SVM, XGboost, Random Forest should be able to give best results in test set evaluation.

4. **Following table shows performances of models on test set after PCA transformation**:

| Models | Accuracy | F1-score | Precision | Recall |
|---|---|---|---|---|
| XG Boost Classifier | 0.99918 | 0.99916 | 1.00000 | 0.99833 |
| Support Vector Machines | 1.00000 | 1.00000 | 1.00000 | 1.00000 |
| Random Forest | 0.99836 | 0.99832 | 1.00000 | 0.99666 |
| AdaBoost Classifier | 0.98359 | 0.98322 | 0.98653 | 0.97993 |
| Decision Tree Classifier | 0.99754 | 0.99750 | 0.99501 | 1.00000 |

For this performance analysis models have been trained on training set with optimized parameters and performance is evaluated on testing set which was kept in isolation from training. Above are the results which are obtained

## Justification

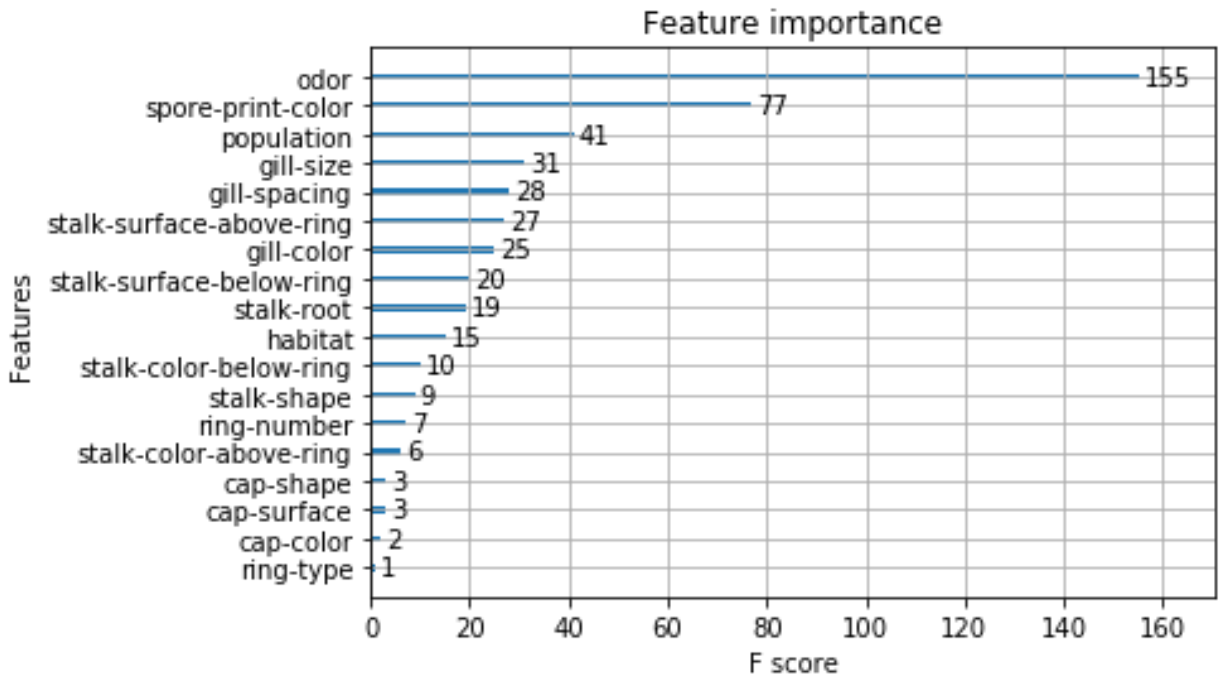
Final performance evaluation obtained from testing the trained models with isolated testing data show that the models have well surpassed the benchmark provided. Average performances for all the metrics for benchmark model is about 0.6 and average performance for all the metrics for our best model(SVM) is 1.0 and rest of the models are between 0.98 to 1.0.

It can be concluded that for this project SVM should be the best model for training in reduced dimensions as it can completely classify all the samples correctly and since testing data was kept isolated, it can also be concluded that it generalizes well for unseen data. And as seen above that SVM had consistently high-performance values regardless if the dimensionality has been reduced or not. So, it will be robust to any minor changes in training data or dimensionality in future. Along with perfect precision score, it has perfect performance for F1-score which signifies that there is complete harmony between precision and recall. Perfect performance in accuracy denotes that it correctly classifies all the samples weather it is edible or poisonous. Thus, for all these reasons the model can be trusted for classification.

Regarding other models although rest of the models also have precision more than 0.98 they might sometimes misclassify since recall and accuracy are not 1.0. Thus if precision is the sole decider than along with SVM, XGBoost and Random Forest could also be used for classification. The only drawback of SVM is that it takes more time for training.

# V. Conclusion

## Free-Form Visualization




The above plot shows feature importance of the dataset. It can be seen that odor is the most important feature of the dataset. In fact odor is the determining factor if a food substance is poisonous or edible. Hence it is within expectations that it is the most important. Odor could be said to be common feature for determining if a food item is edible or not. And it can be found in almost all classification type problems of food items. Another thing about the data set is that it is categorical in nature so each feature would have some fixed number of values associated with it so models that use trees are normally used more. Few of the last features can be said to be of least importance and do not contribute much to the classification. So dimensionality reduction can be used which will in turn make training models faster and reduce computation cost.

## Reflection

Project started with finding dataset for classification which was found on Kaggle. Followed by data visualization and analysis where in it number of attributes and samples was found. An important characteristic of the dataset is that it is completely made of categorical values. This increased the difficulty for processing data and training as number of models use only numeric values. So in order to ensure that most of the models could be used it was decided to have the features encoded. After which the data was split into samples and labels and then encoded into numeric values by using LabelEncoder. This concluded the data preprocessing part.

Next came the initial training and analysis part. In this part the data was split into training , validation and testing sets in the ratio 14:3:3(70% ,15%,15%) respectively. Models were trained on training sets and performance evaluated on validation set. It was found that all the models did considerably well except for Logistic Regression and Stochastic Gradient descent. Thus there arose a need to analyze the features of dataset and decide on dimension reduction. Analysis was done on the features with the help of PCA and dataset was reduced to 16 principle components as any more did not contribute much to the data variance. It was interesting to see that components 17-22 only contributed to about 0.5% of the data variance. The models were again trained and tested on the transformed dataset. Result showed that performance dropped for all the models slightly except for Logistic Regression and Stochastic Gradient descent whose precision score went below 0.90. They were left out from further processing.

Then came optimization part. In this part, certain parameters were defined for each of the models and were provided to Grid Search CV along with the model and scoring function which was precision. The output provided the best parameters from the set of given parameters which have the highest precision score. Almost all the models had a very high score.

Then came final performance evaluation on the testing set which was kept isolated up until now. Results showed that SVM is the best classification model as it has been consistently getting outstanding performance results. Models Random Forest and XGBoost followed right after SVM and could be said to also have very good performance. Thus, SVM is chosen as the most appropriate classification model. The model performance exceeded the expectations as it correctly classified all the samples. And, these type of binary classification problems which have small dataset and are not time constraint can be easily solved using SVM.

## Improvement

There are many aspects where improvement could be made on this project. Few of them are listed below:

1. Gathering additional data for all poisonous substances and make a general classifier which classifies weather an object is poisonous or not.
2. Use of neural networks and reinforcement learning provided huge amount of data availability. And in those cases, models like Logistic Regression and Stochastic Gradient Descent should work well.
3. Parallelization of parameter optimization process for all the models.
4. Training time analysis and cost reduction for training of models.
5. Using final solution as the benchmark innovative solutions could be made where in time it takes for training is drastically reduced. It could also include batch processing.

## Sources


https://www.kaggle.com

http://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning

https://en.wikipedia.org

http://scikit-learn.org

https://xgboost.readthedocs.io

https://www.cs.cornell.edu/~caruana/ctp/ct.papers/caruana.icml06.pdf