MLND Capstone Project

Predict Wine Class

Definition

Project Overview

Wine tasting is the sensory examination and evaluation of wine. While the practice of wine tasting is as ancient as its production, a more formalized methodology has slowly become established from the 14th century onwards. Modern, professional wine tasters (such as sommeliers or buyers for retailers) use a constantly evolving specialized terminology which is used to describe the range of perceived flavours, aromas and general characteristics of a wine. More informal, recreational tasting may use similar terminology, usually involving a much less analytical process for a more general, personal appreciation. (Source: Wikipedia)

One of the goal of AI/ML algorithms is to reduce dependency on human intelligence and develop scientific methods for decision making. Here we will be exploring various algorithm to develop model to predict wine class scientifically.

In this project we are going to analyse the data, which is the result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The aim is to develop a model to predict class of wine based on 13 parameters which is result of laboratory test of the wine. As per UCI website the original dataset had 30 attributes but available dataset has 30 attributes.
Please find below few links for details of research papers on multiclass classification:

https://en.wikipedia.org/wiki/Multiclass_classification

http://scikit-learn.org/stable/modules/multiclass.html

http://www.mit.edu/~9.520/spring09/Classes/multiclass.pdf

https://www.cs.utah.edu/~piyush/teaching/aly05multiclass.pdf

Problem Statement

It is a typical classification problem where we have to predict class based on the given attributes/features, I will explore various classifiers at predicting a certain wine type from available UCI's wine dataset.

The initial data set had around 30 variables, but on UCI's website only 13-dimensional version is available. (All attributes are continuous)

1) Alcohol	6) Total phenols	11) Hue
2) Malic acid	7) Flavanoids	12) OD280/OD315 of diluted wines
3) Ash	8) Nonflavanoid phenols	13) Proline
4) Alcalinity of ash	9) Proanthocyanins	
5) Magnesium	10) Colour intensity	

In a classification context, this is a well posed problem with "well behaved" class structures. Number of Instances: class-1(59), class-2(71), class-3(48), Total 178 records in csv format.
https://archive.ics.uci.edu/ml/machine-learning-databases/wine/

For this problem I would first explore the data and find out if there is any anomalies in the data and visualize all the attributes to check characteristics of the data. If most the attributes have Gaussian distribution then I would also try standard scaler transformation to get better result from classification model.

I am going to try most popular linear, non linear and ensemble algorithms for multiclass classification. I would first try algorithm with default parameter to check which one is more suitable for given problem. After narrowing down the top 2 algorithms I would tune the parameter to improve the accuracy.

Metrics

I will use accuracy and scikit-learn Classification Report as evaluation matrix which reports precision, recall and f1-score.

Accuracy measures how often the classifier makes the correct prediction. It's the ratio of the number of correct predictions to the total number of predictions (the number of test data points).

Precision tells us what proportion of data we classified as particular class were actually from particular class. It is a ratio of true positives to all of positives irrespective of whether that was the correct classification), in other words it is the ratio of True Positives to (True Positives + False Positives).

Recall(sensitivity) tells us what proportion of messages that actually were from that class were classified by model as that particular class. It is a ratio of true positives of that class to all the data belong to that class, in other words it is the ratio of True Positives to (True Positives + False Negatives).

For multiclass classification problems accuracy by itself is not a very good metric. for example if we have 100 data point with 40 of class-1 30 from class-2 and 30 from class-3 if the model marks all the class one and to class two correctly and marks all the class-3 data to class-1 or class-2 even then it will have 70% accuracy but actually it can not identify class-3 at all. For such cases, precision and recall come in very handy. These two metrics can be combined to get the F1 score, which is weighted average(harmonic mean) of the precision and recall scores. This score can range from 0 to 1, with 1 being the best possible F1 score(we take the harmonic mean as we are dealing with ratios).

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Analysis

Data Exploration

Index	Description	Data Type
0	Wine Class (1, 2, 3)	int64
1	Alcohol	float64
2	Malic acid	float64
3	Ash	float64
4	Alcalinity of ash	float64
5	Magnesium	int64
6	Total phenols	float64

7	Flavanoids	float64
8	Nonflavanoid phenols	float64
9	Proanthocyanins	float64
10	Colour intensity	float64
11	Hue	float64
12	OD280/OD315 of diluted wines	float64
13	Proline	int64

Sample data

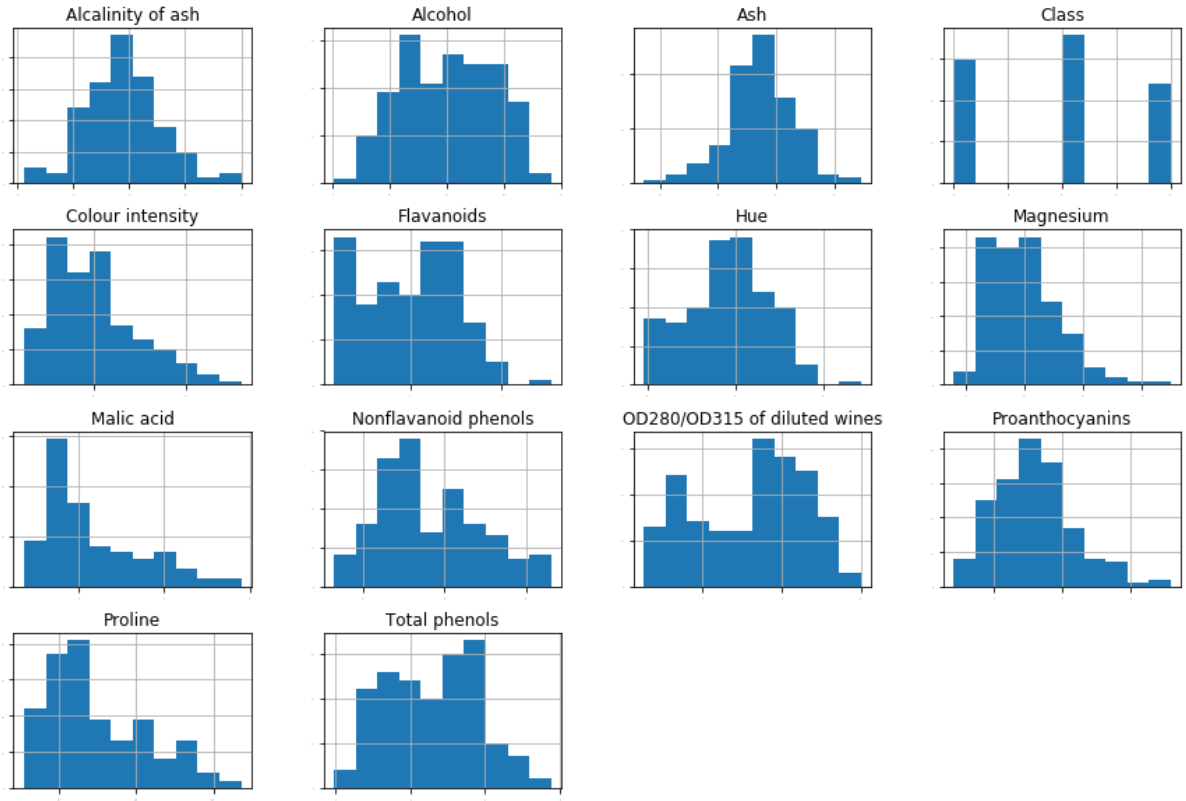
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
5	1	14.20	1.76	2.45	15.2	112	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450
6	1	14.39	1.87	2.45	14.6	96	2.50	2.52	0.30	1.98	5.25	1.02	3.58	1290
7	1	14.06	2.15	2.61	17.6	121	2.60	2.51	0.31	1.25	5.05	1.06	3.58	1295
8	1	14.83	1.64	2.17	14.0	97	2.80	2.98	0.29	1.98	5.20	1.08	2.85	1045
9	1	13.86	1.35	2.27	16.0	98	2.98	3.15	0.22	1.85	7.22	1.01	3.55	1045
10	1	14.10	2.16	2.30	18.0	105	2.95	3.32	0.22	2.38	5.75	1.25	3.17	1510
11	1	14.12	1.48	2.32	16.8	95	2.20	2.43	0.26	1.57	5.00	1.17	2.82	1280
12	1	13.75	1.73	2.41	16.0	89	2.60	2.76	0.29	1.81	5.60	1.15	2.90	1320
13	1	14.75	1.73	2.39	11.4	91	3.10	3.69	0.43	2.81	5.40	1.25	2.73	1150
14	1	14.38	1.87	2.38	12.0	102	3.30	3.64	0.29	2.96	7.50	1.20	3.00	1547
15	1	13.63	1.81	2.70	17.2	112	2.85	2.91	0.30	1.46	7.30	1.28	2.88	1310
16	1	14.30	1.92	2.72	20.0	120	2.80	3.14	0.33	1.97	6.20	1.07	2.65	1280
17	1	13.83	1.57	2.62	20.0	115	2.95	3.40	0.40	1.72	6.60	1.13	2.57	1130
18	1	14.19	1.59	2.48	16.5	108	3.30	3.93	0.32	1.86	8.70	1.23	2.82	1680
19	1	13.64	3.10	2.56	15.2	116	2.70	3.03	0.17	1.66	5.10	0.96	3.36	845

Data Description

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
count	178.00	178.00	178.00	178.00	178.00	178.00	178.00	178.00	178.00	178.00	178.00	178.00	178.00	178.00
mean	1.94	13.00	2.34	2.37	19.49	99.74	2.30	2.03	0.36	1.59	5.06	0.96	2.61	746.89
std	0.78	0.81	1.12	0.27	3.34	14.28	0.63	1.00	0.12	0.57	2.32	0.23	0.71	314.91
min	1.00	11.03	0.74	1.36	10.60	70.00	0.98	0.34	0.13	0.41	1.28	0.48	1.27	278.00
25%	1.00	12.36	1.60	2.21	17.20	88.00	1.74	1.20	0.27	1.25	3.22	0.78	1.94	500.50
50%	2.00	13.05	1.87	2.36	19.50	98.00	2.35	2.13	0.34	1.56	4.69	0.96	2.78	673.50
75%	3.00	13.68	3.08	2.56	21.50	107.00	2.80	2.88	0.44	1.95	6.20	1.12	3.17	985.00
max	3.00	14.83	5.80	3.23	30.00	162.00	3.88	5.08	0.66	3.58	13.00	1.71	4.00	1680.00

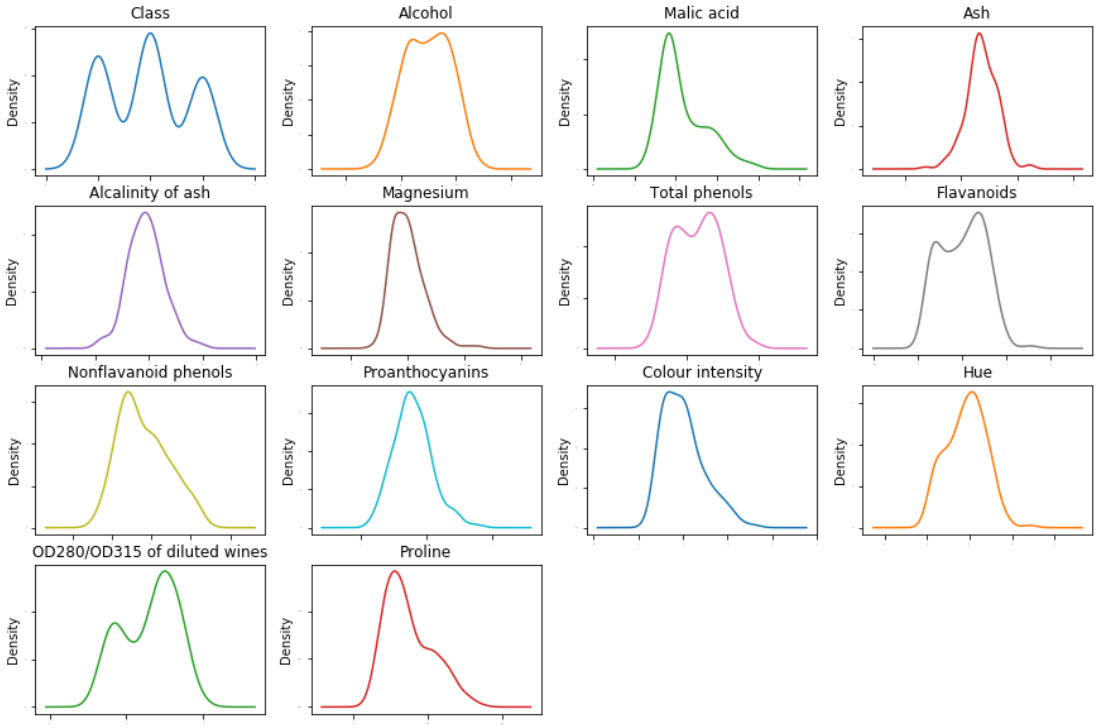
Histogram

We can see that all of the data has distribution close to Gaussian



Density

Density diagram shown few of the features are left or right skewed but most of them follow Gaussian structure. We check the density for checking skewness of data and if data is very skewed then we try Box-Cox transformation.



Data Summary

Number of Records: 178

No of classes: 3

No of Features: 13

Distribution of Data by Class: Class-1(59), Class-2(71), Class-3(48),

Index	Description	Type	Min	Max	Std	Missing data	Structure (Approx)
1	Alcohol	Float	11.03	14.83	0.81	None	Gaussian
2	Malic acid	Float	0.74	5.80	1.12	None	Gaussian – Left Skewed
3	Ash	Float	1.36	3.23	0.27	None	Gaussian – Right Skewed
4	Alcalinity of ash	Float	10.60	30	3.34	None	Gaussian
5	Magnesium	Int	70	162	14.28	None	Gaussian – Left Skewed
6	Total phenols	Float	0.98	3.88	0.63	None	Gaussian
7	Flavanoids	Float	0.34	5.08	1.00	None	Gaussian – Left Skewed
8	Nonflavanoid phenols	Float	0.13	0.66	0.12	None	Gaussian
9	Proanthocyanins	Float	0.41	3.58	0.57	None	Gaussian – Left Skewed
10	Colour intensity	Float	1.28	13.00	2.32	None	Gaussian – Left Skewed
11	Hue	Float	0.48	1.71	0.23	None	Gaussian
12	OD280/OD315 of diluted wines	Float	1.27	4.00	0.71	None	Gaussian – Right Skewed
13	Proline	Int	278	1680	314.91	None	Gaussian – Left Skewed

All of the features have no missing values and most of them have gaussian distribution, which makes it easier for us to analyse and build a predictive model.

Algorithms and Techniques

- ✓ Prepare Problem
  - Load all the required python libraries
  - Load the csv dataset from UCI's website
- ✓ Summarize Data
  - Use Descriptive statistics to get insight of the available data. Total data points, distribution of data class wise, data type, mean, median etc
  - Data visualizations with histogram and density to check characteristic of the data
- ✓ Prepare Data
  - Data Cleaning may not be required as it contains well behaved data
  - Feature Selection - I will be using all the 13 attributes to develop the predictive model

- Data Transforms - I will try StandardScaler transformation of the data and check the algorithms accuracy before and after the transformation.

✓ Evaluate Algorithms
- Split-out the dataset into training and validation set
- Defining test options using scikit-learn such as k-fold cross-validation and the evaluation metric to use.
- Selecting few Linear (Logistic Regression) and Non-Linear Algorithms (KNN, SVM, Decision Tree, Naïve Bayes) and do the initial check with default parameters.
- The algorithms all use default tuning parameters. We will display the mean and standard deviation of accuracy for each algorithm as we calculate it and collect the results for use later.
- Once we get the best fit algorithm for our problem we will try to tune the parameters for best result.

✓ Improve Accuracy
- Tuning of the selected Algorithm to achieve highest accuracy
- Trying Ensembles (Gradient Boosting, Random Forest, Ada Boost, Extra Trees) to improve accuracy

✓ Finalize Model
- Predictions on validation dataset using the best algorithm chosen from above steps
- Create standalone model on entire training dataset
- Save model for later use.

The biggest problem was choosing algorithm for this problem . That's why I decide to all algorithms from each category like linear classification algorithms (Logistic Regression), Non-Linear Classification algorithms (SVM, Decision Tree, Naïve Bayes), unsupervised classification algorithms like KNN, Ensembles like Gradient Boosting, Random Forest, Ada Boost, Extra Trees. And tuned all the top algorithms from each category. Because choosing the correct algorithm takes extensive experience in machine learning and I could not decide which one would work best for me. Since I had small amount of data so I had flexibility to try all the algorithms I believe may suit the problem and tune the winner. May be after working on Machine Learning for significant time I would develop gut feeling to decide which algorithm would work for given type of problem.

Here is the description of top 3 algorithms I chose for this problem.

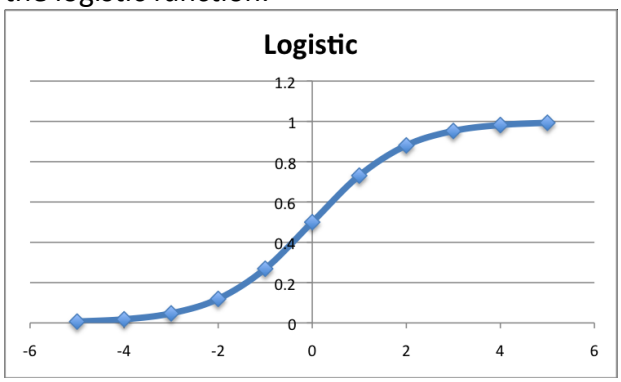
Logistics Regression:

Logistic regression is named for the function used at the core of the method, the logistic function. The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

$$y = \frac{e^{B0+B1\times x}}{1+e^{B0+B1\times x}}$$

Where y is the predicted output, B0 is the bias or intercept term and B1 is the coefficient for the single input value (x). Each column in your input data has an associated B coefficient (a constant real value) that must be learned from your training data. The actual representation of the model that you would store in memory or in a file are the coefficients in the equation (the beta value or B's).

Below is a plot of the numbers between -5 and 5 transformed into the range 0 and 1 using the logistic function.



SVM

The SVM algorithm is implemented in practice using a kernel. The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra. A powerful insight is that the linear SVM can be rephrased using the inner product of any two given observations, rather than the observations themselves.

The inner product between two vectors is the sum of the multiplication of each pair of input values. For example, the inner product of the vectors [2; 3] and [5; 6] is 2 _ 5 + 3 _ 6 or 28.

The equation for making a prediction for a new input using the dot product between the input (x) and each support vector (x_i) is calculated as follows:

$$f(x) = B0 + \sum_{i=1}^{n}(a_i \times (x \times x_i))$$

This is an equation that involves calculating the inner products of a new input vector (x) with all support vectors in training data. The coefficients B0 and a_i (for each input) must be estimated from the training data by the learning algorithm.

The kernel defines the similarity or a distance measure between new data and the support vectors. The dot product is the similarity measure used for linear SVM or a linear kernel because the distance is a linear combination of the inputs. Other kernels can be used that transform the input space into higher dimensions such as a Polynomial Kernel and a Radial Kernel. This is called the Kernel Trick. It is desirable to use more complex kernels as it allows lines to separate the classes that are curved or even more complex. This in turn can lead to more accurate classifiers.

GradientBoosting

Stochastic Gradient Boosting (also called Gradient Boosting Machines) are one of the most sophisticated ensemble techniques. It is also a technique that is proving to be perhaps one of the best techniques available for improving performance via ensembles. You can construct a Gradient Boosting model for classi_cation using the GradientBoostingClassifier class of sklearn.

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

Algorithm [edit]

In many supervised learning problems one has an output variable y and a vector of input variables x connected via a joint probability distribution $P(x, y)$. Using a training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ of known values of x and corresponding values of y, the goal is to find an approximation $\hat{F}(x)$ to a function $F(x)$ that minimizes the expected value of some specified loss function $L(y, F(x))$:

$$\hat{F} = \arg\min_F \mathbb{E}_{x,y}[L(y, F(x))].$$

The gradient boosting method assumes a real-valued y and seeks an approximation $\hat{F}(x)$ in the form of a weighted sum of functions $h_i(x)$ from some class $\mathcal{H}$, called base (or weak) learners:

$$F(x) = \sum_{i=1}^{M} \gamma_i h_i(x) + \text{const}.$$

In accordance with the empirical risk minimization principle, the method tries to find an approximation $\hat{F}(x)$ that minimizes the average value of the loss function on the training set. It does so by starting with a model, consisting of a constant function $F_0(x)$, and incrementally expanding it in a greedy fashion:

$$F_0(x) = \arg\min_\gamma \sum_{i=1}^{n} L(y_i, \gamma),$$

$$F_m(x) = F_{m-1}(x) + \arg\min_{h_m \in \mathcal{H}} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h_m(x_i)),$$

where $h_m \in \mathcal{H}$ is a base learner function.

Unfortunately, choosing the best function h at each step for an arbitrary loss function L is a computationally infeasible optimization problem in general. Therefore, we will restrict to a simplification.

The idea is to apply a steepest descent step to this minimization problem. If we considered the continuous case, i.e. $\mathcal{H}$ the set of arbitrary differentiable functions on $\mathbb{R}$, we would update the model in accordance with the following equations

$$F_m(x) = F_{m-1}(x) - \gamma_m \sum_{i=1}^{n} \nabla_{F_{m-1}} L(y_i, F_{m-1}(x_i)),$$

$$\gamma_m = \arg\min_\gamma \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) - \gamma \nabla_{F_{m-1}} L(y_i, F_{m-1}(x_i))\right),$$

where the derivatives are taken with respect to the functions F_i for $i \in \{1, \ldots, m\}$. In the discrete case however, i.e. the set $\mathcal{H}$ is finite, we will choose the candidate function h closest to the gradient of L for which the coefficient γ may then be calculated with the aid of line search the above equations. Note that this approach is a heuristic and will therefore not yield an exact solution to the given problem, yet a satisfactory approximation.

Source:Wiki

Benchmark

I will use following result on Kaggle as benchmark:
https://www.kaggle.com/brynja/testing-classifiers-for-wine-variety-prediction

Accuracy: 0.98592

Sensitivity: 0.95455

Specificity: 1.00000

Precision: 1.00000

I will try tune my algorithms and try few ensembles to improve accuracy and get better result than above mention model. However if I talk in term of independent project I would keep LogisticRegression as benchmark model and would compare other model with it for benchmarking the result.

Methodology

Data Pre-processing

The given data is well processed data having no anomalies, none of the attributed had null or garbage values. And we have good number of samples from each class. The only preprocessing I did on data is doing StandardScaler transform.

Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1. It is most suitable for techniques that assume a Gaussian distribution in the input variables and work better with rescaled data.

Data

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	12.96	3.45	2.35	18.5	106.0	1.39	0.70	0.40	0.94	5.28	0.68	1.75	675.0
1	13.71	5.65	2.45	20.5	95.0	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740.0
2	14.12	1.48	2.32	16.8	95.0	2.20	2.43	0.26	1.57	5.00	1.17	2.82	1280.0
3	13.58	2.58	2.69	24.5	105.0	1.55	0.84	0.39	1.54	8.66	0.74	1.80	750.0
4	13.30	1.72	2.14	17.0	94.0	2.40	2.19	0.27	1.35	3.95	1.02	2.77	1285.0

Rescaled Data

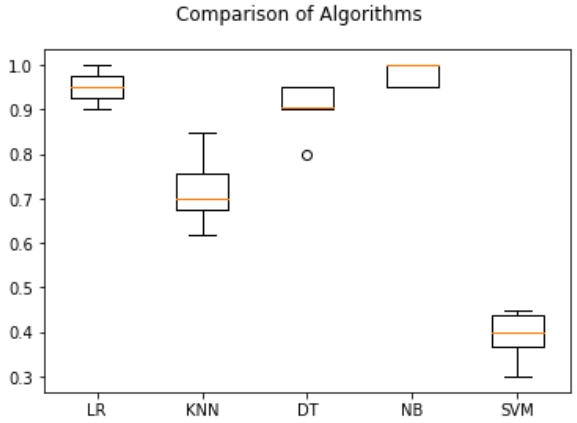
	0	1	2	3	4	5	6	7	8	9	10	11	12
0	-0.11	0.99	-0.06	-0.27	0.40	-1.42	-1.35	0.32	-1.13	0.07	-1.27	-1.24	-0.30
1	0.83	2.94	0.30	0.32	-0.37	-0.97	-1.44	1.27	-0.92	1.17	-1.45	-1.25	-0.10
2	1.34	-0.76	-0.16	-0.77	-0.37	-0.17	0.36	-0.80	-0.04	-0.06	0.94	0.29	1.56
3	0.66	0.21	1.15	1.49	0.33	-1.18	-1.21	0.24	-0.09	1.61	-1.00	-1.17	-0.07
4	0.31	-0.55	-0.80	-0.71	-0.44	0.14	0.13	-0.72	-0.42	-0.53	0.26	0.22	1.57

Implementation

Firstly, I tried few algorithms to check which one is performing better with default parameters. I tried Linear Classification model (Logistics Regression) and non-Linear classification models (KNN, SVM, Decision Tree, Naïve Bayes. Here is the result (Mean accuracy and standard deviation):

```
Logistic Regression: 0.950680 (0.037812)
KNN: 0.718707 (0.071149)
Decision Tree: 0.908163 (0.049713)
Naïve Bayes: 0.978571 (0.024744)
SVM: 0.394218 (0.051332)
```

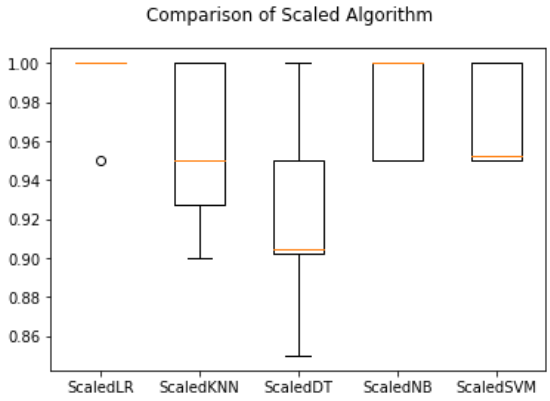
As per the result Naive Bayes is performing best out of all the 5 algorithms. Logistic Regression and Decision Tree are also very close to NB.



I tried all these algorithms again with transformed data and here is the result:

```
ScaledLR: 0.992857 (0.017496)
```

```
ScaledKNN: 0.957823 (0.040740)
ScaledDT: 0.922789 (0.044577)
ScaledNB: 0.978571 (0.024744)
ScaledSVM: 0.971769 (0.024461)
```



Clearly Logistic Regression is the winner.

Refinement

I tuned parameters of Logistic Regression and SVM using grid search to get the best result out of it.

Here is the result of tuning:

```
Logistic Regression: 0.992958 using {'solver': 'lbfgs', 'C': 1.0}
```

```
Best: 0.992958 using {'solver': 'lbfgs', 'C': 1.0}
0.971831 (0.036089) with: {'solver': 'lbfgs', 'C': 0.01}
0.971831 (0.036089) with: {'solver': 'newton-cg', 'C': 0.01}
0.971831 (0.036089) with: {'solver': 'sag', 'C': 0.01}
0.978873 (0.024356) with: {'solver': 'lbfgs', 'C': 0.1}
0.978873 (0.024356) with: {'solver': 'newton-cg', 'C': 0.1}
0.978873 (0.024356) with: {'solver': 'sag', 'C': 0.1}
0.992958 (0.017393) with: {'solver': 'lbfgs', 'C': 1.0}
0.992958 (0.017393) with: {'solver': 'newton-cg', 'C': 1.0}
0.992958 (0.017393) with: {'solver': 'sag', 'C': 1.0}
0.985915 (0.022491) with: {'solver': 'lbfgs', 'C': 20.0}
0.985915 (0.022491) with: {'solver': 'newton-cg', 'C': 20.0}
0.985915 (0.022491) with: {'solver': 'sag', 'C': 20.0}
0.978873 (0.024698) with: {'solver': 'lbfgs', 'C': 500.0}
0.978873 (0.024698) with: {'solver': 'newton-cg', 'C': 500.0}
0.985915 (0.022491) with: {'solver': 'sag', 'C': 500.0}
```

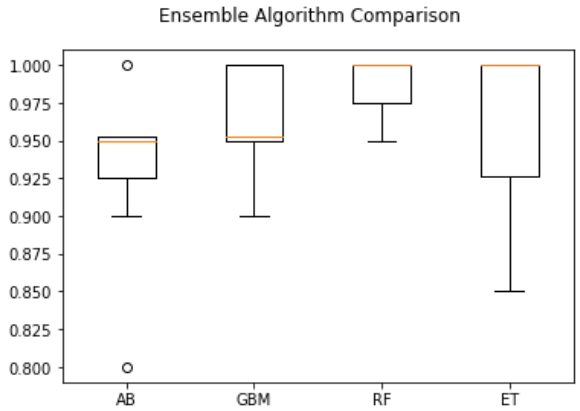
```
SVM : 0.992958 using {'C': 0.1, 'kernel': 'linear'}
```

```
Best: 0.992958 using {'C': 0.1, 'kernel': 'linear'}
0.992958 (0.017393) with: {'C': 0.1, 'kernel': 'linear'}
0.802817 (0.079520) with: {'C': 0.1, 'kernel': 'poly'}
0.964789 (0.034028) with: {'C': 0.1, 'kernel': 'rbf'}
0.971831 (0.024458) with: {'C': 0.1, 'kernel': 'sigmoid'}
0.957746 (0.017140) with: {'C': 0.3, 'kernel': 'linear'}
0.894366 (0.049394) with: {'C': 0.3, 'kernel': 'poly'}
0.971831 (0.024458) with: {'C': 0.3, 'kernel': 'rbf'}
0.985915 (0.022115) with: {'C': 0.3, 'kernel': 'sigmoid'}
0.964789 (0.022449) with: {'C': 0.5, 'kernel': 'linear'}
0.929577 (0.035278) with: {'C': 0.5, 'kernel': 'poly'}
0.971831 (0.024458) with: {'C': 0.5, 'kernel': 'rbf'}
0.985915 (0.022115) with: {'C': 0.5, 'kernel': 'sigmoid'}
0.971831 (0.024458) with: {'C': 0.7, 'kernel': 'linear'}
0.957746 (0.031591) with: {'C': 0.7, 'kernel': 'poly'}
0.971831 (0.024458) with: {'C': 0.7, 'kernel': 'rbf'}
0.985915 (0.022115) with: {'C': 0.7, 'kernel': 'sigmoid'}
0.978873 (0.024356) with: {'C': 0.9, 'kernel': 'linear'}
0.957746 (0.031591) with: {'C': 0.9, 'kernel': 'poly'}
0.971831 (0.024458) with: {'C': 0.9, 'kernel': 'rbf'}
0.985915 (0.022115) with: {'C': 0.9, 'kernel': 'sigmoid'}
0.978873 (0.024356) with: {'C': 1.0, 'kernel': 'linear'}
0.957746 (0.031591) with: {'C': 1.0, 'kernel': 'poly'}
0.971831 (0.024458) with: {'C': 1.0, 'kernel': 'rbf'}
0.985915 (0.022115) with: {'C': 1.0, 'kernel': 'sigmoid'}
0.978873 (0.024356) with: {'C': 1.3, 'kernel': 'linear'}
0.971831 (0.036089) with: {'C': 1.3, 'kernel': 'poly'}
0.978873 (0.024356) with: {'C': 1.3, 'kernel': 'rbf'}
0.978873 (0.024356) with: {'C': 1.3, 'kernel': 'sigmoid'}
0.985915 (0.022115) with: {'C': 1.5, 'kernel': 'linear'}
0.971831 (0.036089) with: {'C': 1.5, 'kernel': 'poly'}
0.978873 (0.024356) with: {'C': 1.5, 'kernel': 'rbf'}
0.971831 (0.036089) with: {'C': 1.5, 'kernel': 'sigmoid'}
0.985915 (0.022115) with: {'C': 1.7, 'kernel': 'linear'}
0.971831 (0.036089) with: {'C': 1.7, 'kernel': 'poly'}
0.978873 (0.024356) with: {'C': 1.7, 'kernel': 'rbf'}
0.971831 (0.024113) with: {'C': 1.7, 'kernel': 'sigmoid'}
0.985915 (0.022115) with: {'C': 2.0, 'kernel': 'linear'}
0.971831 (0.036089) with: {'C': 2.0, 'kernel': 'poly'}
0.978873 (0.024356) with: {'C': 2.0, 'kernel': 'rbf'}
0.964789 (0.022073) with: {'C': 2.0, 'kernel': 'sigmoid'}
```

I tried ensembles to see if I can get better accuracy as compared to LR or SVM. I tried (Gradient Boosting, Random Forest, Ada Boost, Extra Trees) with default parameters to check which one is performing best for our case.

Here is the result:

```
AdaBoost: 0.929252 (0.059158)
GradientBoosting: 0.964626 (0.034864)
RandomForest: 0.985714 (0.022588)
ExtraTrees: 0.957483 (0.056206)
```



I tried to tune both GradientBoosting and RandomForest and got best result from GradientBoosting.

```
Best: 1.000000 using {'n_estimators': 10, 'min_samples_split': 6, 'max_
features': 3, 'min_samples_leaf': 1, 'max_depth': 3}
```

I did not face any issue with the coding as I have done various projects on classification as part of machine learning nano degree and self driving car nano degree and I have also done few projects on my own to practice what I have learnt.

Results

Model Evaluation and Validation

After following all the above steps GradientBoosting model was a clear winner. The model seems to give ideal result as 1 for accuracy, precision, recall and f1-score. This could be because we have small validation set but we can consider this as fairly accurate model. It identified all the data class perfectly with no error. The validation dataset had 36 data point with 7 from class 1, 17 from class-2 and 12 from class-3 and model identified all the data correctly, therefore having score of 1 in precision, recall and F-1.

Model Robustness

```
1.0
[[ 7  0  0]
 [ 0 17  0]
 [ 0  0 12]]
             precision    recall  f1-score   support

        1.0       1.00      1.00      1.00         7
        2.0       1.00      1.00      1.00        17
        3.0       1.00      1.00      1.00        12

avg / total       1.00      1.00      1.00        36
```

To check the robustness of model I tried validating the model with k-fold cross validation. I tried 7 fold and 9 fold and measured accuracy and f1-score. The model produced really good and robust results for each fold. Please find below the result. It had mean accuracy >98% and F1-Score > 97%, which I think is a great score and very robust model.

	Accuracy		F1-Score	
	Mean	**Standard Deviation**	**Mean**	**Standard Deviation**
7-Fold	.992857	.017496	0.973077	.032692
9-Fold	.986111	.025934	.994302	.016116

```
In [27]: num_folds = 7
         seed = 7
         scoring = 'accuracy'
         kfold = KFold(n_splits=num_folds, random_state=seed)
         cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
         print (cv_results)
         msg = "K Fold Score : %f (%f)" % (cv_results.mean(), cv_results.std())
         print(msg)

         [ 1.    1.    1.    1.    0.95  1.    1.  ]
         K Fold Score : 0.992857 (0.017496)
```

In [29]:
```
num_folds = 9
seed = 9
scoring = 'accuracy'
kfold = KFold(n_splits=num_folds, random_state=seed)
cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
print (cv_results)
msg = "K Fold Score : %f (%f)" % (cv_results.mean(), cv_results.std())
print(msg)
```

```
[ 1.      1.      1.      1.      1.      0.9375  0.9375  1.      1.    ]
K Fold Score : 0.986111 (0.025984)
```

In [45]:
```
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
num_folds = 7
seed = 3
scoring = 'f1_macro'
kfold = KFold(n_splits=num_folds, random_state=seed)
cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
print (cv_results)
msg = "K Fold Score : %f (%f)" % (cv_results.mean(), cv_results.std())
print(msg)
```

```
[ 1.          1.          1.          0.91534392  0.94747475  0.94871795
  1.        ]
K Fold Score : 0.973077 (0.032692)
```

In [46]:
```
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
num_folds = 9
seed = 3
scoring = 'f1_macro'
kfold = KFold(n_splits=num_folds, random_state=seed)
cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
print (cv_results)
msg = "K Fold Score : %f (%f)" % (cv_results.mean(), cv_results.std())
print(msg)
```

```
[ 1.          1.          1.          1.          1.          1.
  0.94871795  1.          1.        ]
K Fold Score : 0.994302 (0.016116)
```

Justification

In comparison to the benchmark model on Kaggle this model had given much better result with 100% accuracy. As I mentioned in previous heading that this perfect accuracy can be because we have small data set to validate the model. I am sure even with bigger dataset it would give satisfactory result.

Conclusion

Free-Form Visualization

Here are the highlights of the model:

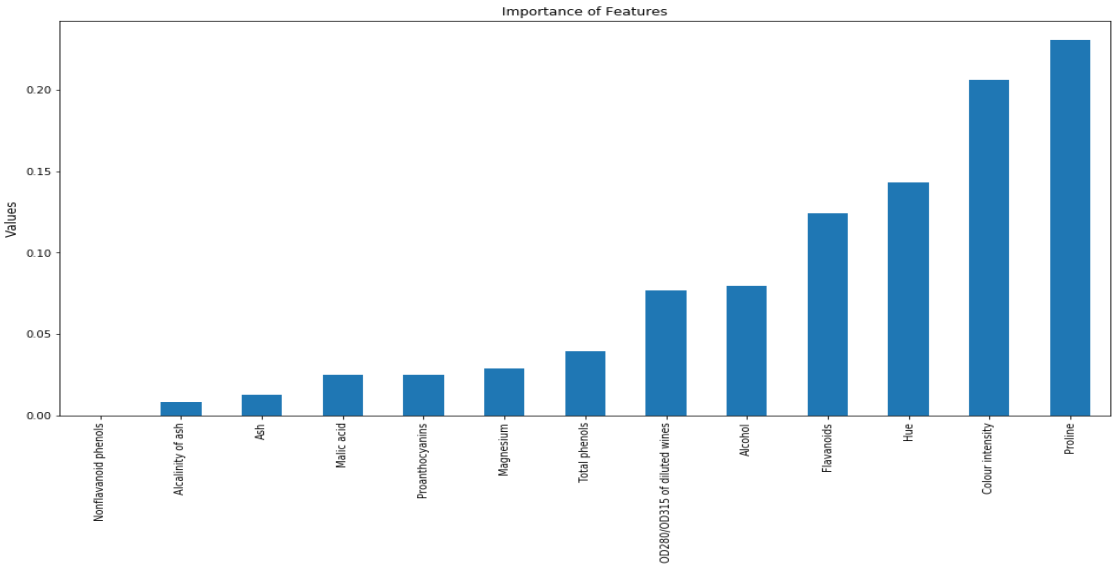
Top 3 and bottom 3 feature as per Importance:

```
Most imp feature = Proline
Least imp feature = Nonflavanoid phenols

3 most imp features of the model:
Proline
Colour intensity
Hue

3 least imp features of the model:
Nonflavanoid phenols
Alcalinity of ash
Ash
```

Plot for importance of features:



I was assuming that first feature (Alcohol) would be the most important feature but it turned out that Feature[12] Proline is the most important feature. And model mostly depend on top 6 features as they carry most of the importance.

Reflection

My solution mainly focussed on optimizing the models by tuning the parameters and in this solution I tried lots of models like Linear model(Logistic Regression), non-Linear classification model (KNN, SVM, Decision Tree, Naïve Bayes) and ensembles (Gradient Boosting, Random Forest, Ada Boost, Extra Trees) to get the best result.

My biggest time consuming tasks in this project was to choose which dataset to choose and approach to adopt. For choosing dataset I decided to go for multiclass classification model as it looked more challenging than linear regression or binary classification problem. I wanted to choose a dataset where I have to do data wrangling and play with data to make to eligible for modelling but could not find suitable data source. My most favourite spot for choosing data for ML is UCI website and I tried various dataset like Sonar Dataset (binary classification), breast cancer data (binary classification), poker hand data (ANN model) and Wine Classification data. Finally I decided to go ahead with data and Kaggle already had a solution for 98% accuracy so getting better accuracy was the challenging task and that's what made me to choose this data set.

For deciding which algorithm to choose I decide to follow the strategy of choosing top 2 algorithms by doing spot check and then tuning the parameter to get best result. I also tried ensemble to better the result.

Improvement

I would definitely like to try this solution with the complete original features (30) and bigger data set. I could also try to include only the important or top 6 features to build the model for more generalize solution.