LOVELY PROFESSIONAL UNIVERSITY

Transforming Education Transforming India

(2018)

INT 246

Project Report

Classification of Iris Dataset

Guided By:

Dr. V. Devendran

Prepared By : Mrinmai Sharma (11603290)
Ravishankar Singh (11610988)

Section : K1632

INDEX

Abstract

This Project involves the use of Iris Data set provided on https://archive.ics.uci.edu/ml/datasets/iris

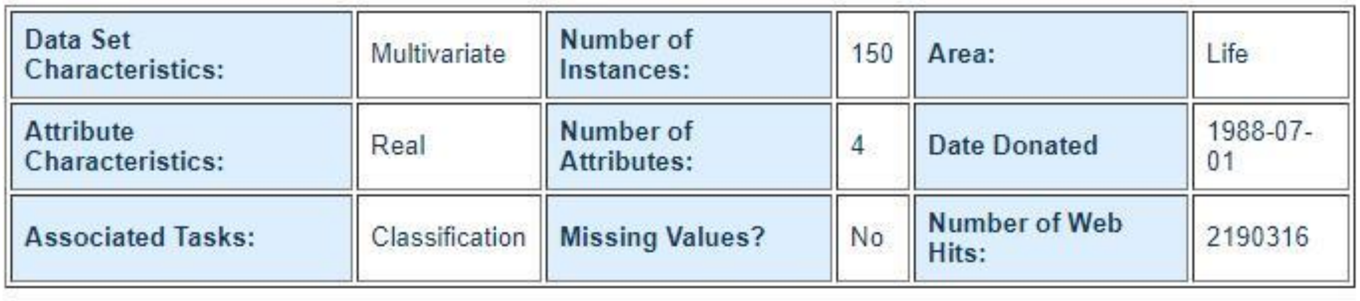
The goal is to summarize the data, provide statistical summary, provide class distribution, visualize the data, build neural network models using various machine learning algorithms, and finally make predictions

About the data set:
This is perhaps the best-known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

Predicted attribute: Class of iris plant.

This is an exceedingly simple domain.

This data differs from the data presented in Fishers article (identified by Steve Chadwick, spchadwick '@' espeedaz.net). The 35th sample should be: 4.9,3.1,1.5,0.2,"Iris-setosa" where the error is in the fourth feature. The 38th sample: 4.9,3.6,1.4,0.1,"Iris-setosa" where the errors are in the second and third features.



Data Set Characteristics:	Multivariate	**Number of Instances:**	150	**Area:**	Life
Attribute Characteristics:	Real	**Number of Attributes:**	4	**Date Donated**	1988-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	2190316

Contributing members of Project:

Mrinmai Sharma (11603290)
Ravishankar Singh (11610988)

Introduction

We are going to use the iris flowers dataset. This dataset is famous because it is used as the “hello world” dataset in machine learning and statistics by pretty much everyone.

The dataset contains 150 observations of iris flowers. There are four columns of measurements of the flowers in centimeters. The fifth column is the species of the flower observed. All observed flowers belong to one of three species.

The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in his 1936 paper The use of multiple measurements in taxonomic problems as an example of linear discriminant analysis. It is sometimes called Anderson's Iris data set because Edgar Anderson collected the data to quantify
the morphologic variation of Iris flowers of three related species. Two of the three species were collected in the Gaspé Peninsula "all from the same pasture, and picked on the same day and measured at the same time by the same person with the same apparatus".

The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

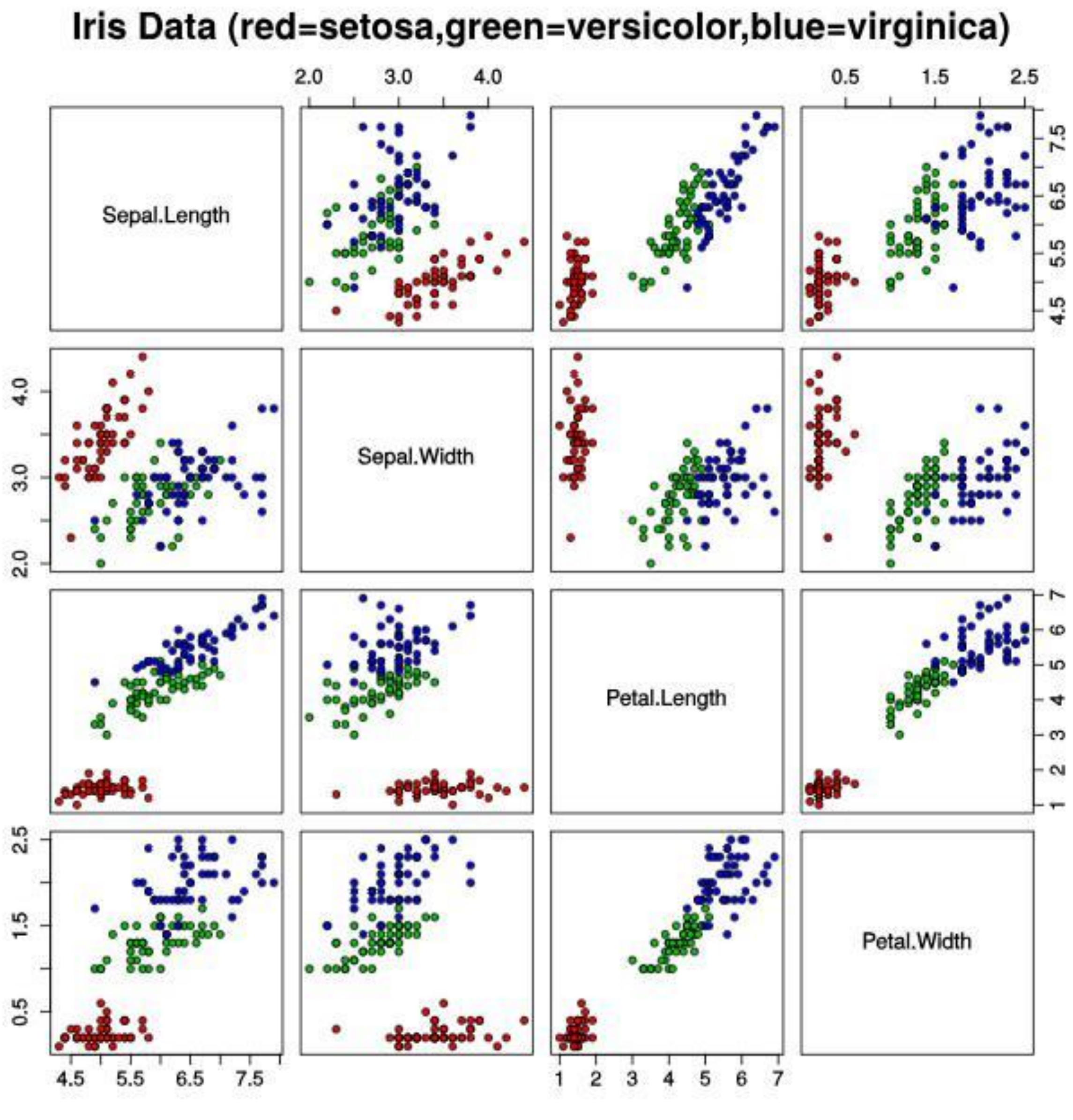
This project makes use of the above-mentioned dataset to create neural network models to provide data analysis and make predictions.

USE OF DATA SET

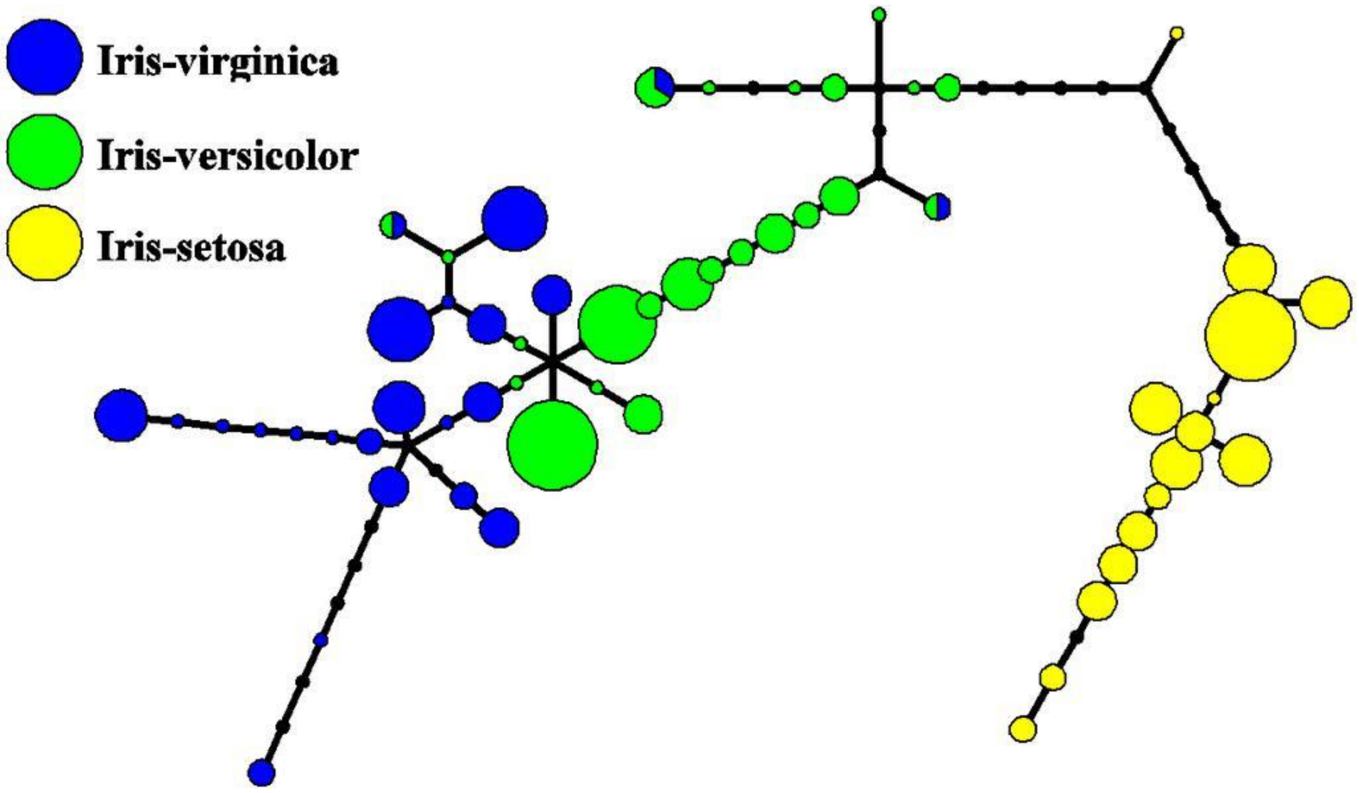
Based on Fisher's linear discriminant model, this data set became a typical test case for many statistical classification techniques in machine learning such as support vector machines.

The use of this data set in cluster analysis however is not common, since the data set only contains two clusters with rather obvious separation. One of the clusters contains Iris setosa, while the other cluster contains both Iris virginica and Iris versicolor and is not separable without the species information Fisher used. This makes the data set a good example to explain the difference between supervised and unsupervised techniques in data mining: Fisher's linear discriminant model can only be obtained when the object species are known: class labels and clusters are not necessarily the same.

Nevertheless, all three species of Iris are separable in the projection on the nonlinear branching principal component. The data set is approximated by the closest tree with some penalty for the excessive number of nodes, bending and stretching. Then the so-called "metro map" is constructed. The data points are projected into the closest node. For each node the pie diagram of the projected points is prepared. The area of the pie is proportional to the number of the projected points. It is clear from the diagram that the absolute majority of the samples of the different Iris species belong to the different nodes. Only a small fraction of Iris-virginica is mixed with Iris-versicolor (the mixed blue-green nodes in the diagram). Therefore, the three species of Iris (Iris setosa, Iris virginica and Iris versicolor) are separable by the un-supervising procedures of nonlinear principal component analysis. To discriminate them, it is sufficient just to select the corresponding nodes on the principal tree.



Non Linear Approach in multidimensional plane



THE DATA SET

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
1	5.1	3.5	1.4	0.2	*I. setosa*
2	4.9	3.0	1.4	0.2	*I. setosa*
3	4.7	3.2	1.3	0.2	*I. setosa*
4	4.6	3.1	1.5	0.2	*I. setosa*
5	5.0	3.6	1.4	0.3	*I. setosa*
6	5.4	3.9	1.7	0.4	*I. setosa*
7	4.6	3.4	1.4	0.3	*I. setosa*
8	5.0	3.4	1.5	0.2	*I. setosa*

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
9	4.4	2.9	1.4	0.2	*I. setosa*
10	4.9	3.1	1.5	0.1	*I. setosa*
11	5.4	3.7	1.5	0.2	*I. setosa*
12	4.8	3.4	1.6	0.2	*I. setosa*
13	4.8	3.0	1.4	0.1	*I. setosa*
14	4.3	3.0	1.1	0.1	*I. setosa*
15	5.8	4.0	1.2	0.2	*I. setosa*
16	5.7	4.4	1.5	0.4	*I. setosa*
17	5.4	3.9	1.3	0.4	*I. setosa*
18	5.1	3.5	1.4	0.3	*I. setosa*
19	5.7	3.8	1.7	0.3	*I. setosa*
20	5.1	3.8	1.5	0.3	*I. setosa*
21	5.4	3.4	1.7	0.2	*I. setosa*
22	5.1	3.7	1.5	0.4	*I. setosa*
23	4.6	3.6	1.0	0.2	*I. setosa*
24	5.1	3.3	1.7	0.5	*I. setosa*
25	4.8	3.4	1.9	0.2	*I. setosa*
26	5.0	3.0	1.6	0.2	*I. setosa*

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
27	5.0	3.4	1.6	0.4	*I. setosa*
28	5.2	3.5	1.5	0.2	*I. setosa*
29	5.2	3.4	1.4	0.2	*I. setosa*
30	4.7	3.2	1.6	0.2	*I. setosa*
31	4.8	3.1	1.6	0.2	*I. setosa*
32	5.4	3.4	1.5	0.4	*I. setosa*
33	5.2	4.1	1.5	0.1	*I. setosa*
34	5.5	4.2	1.4	0.2	*I. setosa*
35	4.9	3.1	1.5	0.2	*I. setosa*
36	5.0	3.2	1.2	0.2	*I. setosa*
37	5.5	3.5	1.3	0.2	*I. setosa*
38	4.9	3.6	1.4	0.1	*I. setosa*
39	4.4	3.0	1.3	0.2	*I. setosa*
40	5.1	3.4	1.5	0.2	*I. setosa*
41	5.0	3.5	1.3	0.3	*I. setosa*
42	4.5	2.3	1.3	0.3	*I. setosa*
43	4.4	3.2	1.3	0.2	*I. setosa*
44	5.0	3.5	1.6	0.6	*I. setosa*

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
45	5.1	3.8	1.9	0.4	*I. setosa*
46	4.8	3.0	1.4	0.3	*I. setosa*
47	5.1	3.8	1.6	0.2	*I. setosa*
48	4.6	3.2	1.4	0.2	*I. setosa*
49	5.3	3.7	1.5	0.2	*I. setosa*
50	5.0	3.3	1.4	0.2	*I. setosa*
51	7.0	3.2	4.7	1.4	*I. versicolor*
52	6.4	3.2	4.5	1.5	*I. versicolor*
53	6.9	3.1	4.9	1.5	*I. versicolor*
54	5.5	2.3	4.0	1.3	*I. versicolor*
55	6.5	2.8	4.6	1.5	*I. versicolor*
56	5.7	2.8	4.5	1.3	*I. versicolor*
57	6.3	3.3	4.7	1.6	*I. versicolor*
58	4.9	2.4	3.3	1.0	*I. versicolor*
59	6.6	2.9	4.6	1.3	*I. versicolor*
60	5.2	2.7	3.9	1.4	*I. versicolor*
61	5.0	2.0	3.5	1.0	*I. versicolor*
62	5.9	3.0	4.2	1.5	*I. versicolor*

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
63	6.0	2.2	4.0	1.0	*I. versicolor*
64	6.1	2.9	4.7	1.4	*I. versicolor*
65	5.6	2.9	3.6	1.3	*I. versicolor*
66	6.7	3.1	4.4	1.4	*I. versicolor*
67	5.6	3.0	4.5	1.5	*I. versicolor*
68	5.8	2.7	4.1	1.0	*I. versicolor*
69	6.2	2.2	4.5	1.5	*I. versicolor*
70	5.6	2.5	3.9	1.1	*I. versicolor*
71	5.9	3.2	4.8	1.8	*I. versicolor*
72	6.1	2.8	4.0	1.3	*I. versicolor*
73	6.3	2.5	4.9	1.5	*I. versicolor*
74	6.1	2.8	4.7	1.2	*I. versicolor*
75	6.4	2.9	4.3	1.3	*I. versicolor*
76	6.6	3.0	4.4	1.4	*I. versicolor*
77	6.8	2.8	4.8	1.4	*I. versicolor*
78	6.7	3.0	5.0	1.7	*I. versicolor*
79	6.0	2.9	4.5	1.5	*I. versicolor*
80	5.7	2.6	3.5	1.0	*I. versicolor*

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
81	5.5	2.4	3.8	1.1	*I. versicolor*
82	5.5	2.4	3.7	1.0	*I. versicolor*
83	5.8	2.7	3.9	1.2	*I. versicolor*
84	6.0	2.7	5.1	1.6	*I. versicolor*
85	5.4	3.0	4.5	1.5	*I. versicolor*
86	6.0	3.4	4.5	1.6	*I. versicolor*
87	6.7	3.1	4.7	1.5	*I. versicolor*
88	6.3	2.3	4.4	1.3	*I. versicolor*
89	5.6	3.0	4.1	1.3	*I. versicolor*
90	5.5	2.5	4.0	1.3	*I. versicolor*
91	5.5	2.6	4.4	1.2	*I. versicolor*
92	6.1	3.0	4.6	1.4	*I. versicolor*
93	5.8	2.6	4.0	1.2	*I. versicolor*
94	5.0	2.3	3.3	1.0	*I. versicolor*
95	5.6	2.7	4.2	1.3	*I. versicolor*
96	5.7	3.0	4.2	1.2	*I. versicolor*
97	5.7	2.9	4.2	1.3	*I. versicolor*
98	6.2	2.9	4.3	1.3	*I. versicolor*

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
99	5.1	2.5	3.0	1.1	*I. versicolor*
100	5.7	2.8	4.1	1.3	*I. versicolor*
101	6.3	3.3	6.0	2.5	*I. virginica*
102	5.8	2.7	5.1	1.9	*I. virginica*
103	7.1	3.0	5.9	2.1	*I. virginica*
104	6.3	2.9	5.6	1.8	*I. virginica*
105	6.5	3.0	5.8	2.2	*I. virginica*
106	7.6	3.0	6.6	2.1	*I. virginica*
107	4.9	2.5	4.5	1.7	*I. virginica*
108	7.3	2.9	6.3	1.8	*I. virginica*
109	6.7	2.5	5.8	1.8	*I. virginica*
110	7.2	3.6	6.1	2.5	*I. virginica*
111	6.5	3.2	5.1	2.0	*I. virginica*
112	6.4	2.7	5.3	1.9	*I. virginica*
113	6.8	3.0	5.5	2.1	*I. virginica*
114	5.7	2.5	5.0	2.0	*I. virginica*
115	5.8	2.8	5.1	2.4	*I. virginica*
116	6.4	3.2	5.3	2.3	*I. virginica*

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
117	6.5	3.0	5.5	1.8	*I. virginica*
118	7.7	3.8	6.7	2.2	*I. virginica*
119	7.7	2.6	6.9	2.3	*I. virginica*
120	6.0	2.2	5.0	1.5	*I. virginica*
121	6.9	3.2	5.7	2.3	*I. virginica*
122	5.6	2.8	4.9	2.0	*I. virginica*
123	7.7	2.8	6.7	2.0	*I. virginica*
124	6.3	2.7	4.9	1.8	*I. virginica*
125	6.7	3.3	5.7	2.1	*I. virginica*
126	7.2	3.2	6.0	1.8	*I. virginica*
127	6.2	2.8	4.8	1.8	*I. virginica*
128	6.1	3.0	4.9	1.8	*I. virginica*
129	6.4	2.8	5.6	2.1	*I. virginica*
130	7.2	3.0	5.8	1.6	*I. virginica*
131	7.4	2.8	6.1	1.9	*I. virginica*
132	7.9	3.8	6.4	2.0	*I. virginica*
133	6.4	2.8	5.6	2.2	*I. virginica*
134	6.3	2.8	5.1	1.5	*I. virginica*

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
135	6.1	2.6	5.6	1.4	*I. virginica*
136	7.7	3.0	6.1	2.3	*I. virginica*
137	6.3	3.4	5.6	2.4	*I. virginica*
138	6.4	3.1	5.5	1.8	*I. virginica*
139	6.0	3.0	4.8	1.8	*I. virginica*
140	6.9	3.1	5.4	2.1	*I. virginica*
141	6.7	3.1	5.6	2.4	*I. virginica*
142	6.9	3.1	5.1	2.3	*I. virginica*
143	5.8	2.7	5.1	1.9	*I. virginica*
144	6.8	3.2	5.9	2.3	*I. virginica*
145	6.7	3.3	5.7	2.5	*I. virginica*
146	6.7	3.0	5.2	2.3	*I. virginica*
147	6.3	2.5	5.0	1.9	*I. virginica*
148	6.5	3.0	5.2	2.0	*I. virginica*
149	6.2	3.4	5.4	2.3	*I. virginica*
150	5.9	3.0	5.1	1.8	*I. virginica*

Libraries used

The following libraries were used:

- ✓ pandas
- ✓ matplotlib
- ✓ seaborn
- ✓ sklearn

```
/***********************************************************
# Load libraries
import pandas
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier from
sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
/************************************************************
```

Loading of The Data Set

We are using pandas to load the data. We will also use pandas next to explore the data both with descriptive statistics and data visualization.

```
/*********************************************************
# Load dataset
url = "iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width',
'class'] dataset = pandas.read_csv(url, names=names)
/**************************************************************
```

Summarizing the Dataset

Dimensions of Dataset

We can get a quick idea of how many instances (rows) and how many attributes (columns) the data contains with the shape property.

```
/**************************************
# shape print(dataset.shape)

/****************************************
```

Peek at the Data

Here ‘20’ refers to the first 20 records of the dataset. To get a peek at much more data, the number can be adjusted accordingly in the function used below.

```
/***************************************************

# head print(dataset.head(20))

/*****************************************************
```

Output:

Sr.no.	Sepal Length	Sepal WIdth	Petal Length	Petal Width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	2.9	1.4	0.1	Iris-setosa
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
15	5.7	4.4	1.5	0.4	Iris-setosa
16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	**5.1**	**3.8**	**1.5**	**0.3**	**Iris-setosa**

Statistical Summary

Summary of each attribute.

This includes the count, mean, the min and max values as well as some percentiles.

```
/***********************************************************
# descriptions print(dataset.describe())


/*******************************************************
```

We can see that all of the numerical values have the same scale (centimeters) and similar ranges between 0 and 8 centimeters.

	Sepal Length	Sepal Width	Petal Length	Petal Width
Count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2800000	1.600000	0.300000
50%	5.800000	3.000000	1.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Class Distribution

We can view the number of instances (rows) that belong to each class as an absolute count.

```
/**********************************************************

# class distribution

print(dataset.groupby('class').size())
/*************************************************
```

Each class has the same number of instances (50 or 33% of the dataset).

Class	Instances
Iris-setosa	50
Iris-versicolor	50
Iris-virginica	50

Data Visualization

We will make use of two types of plots:

- ✓ Univariate plots to better understand each attribute.
- ✓ Multivariate plots to better understand the relationships between attributes.

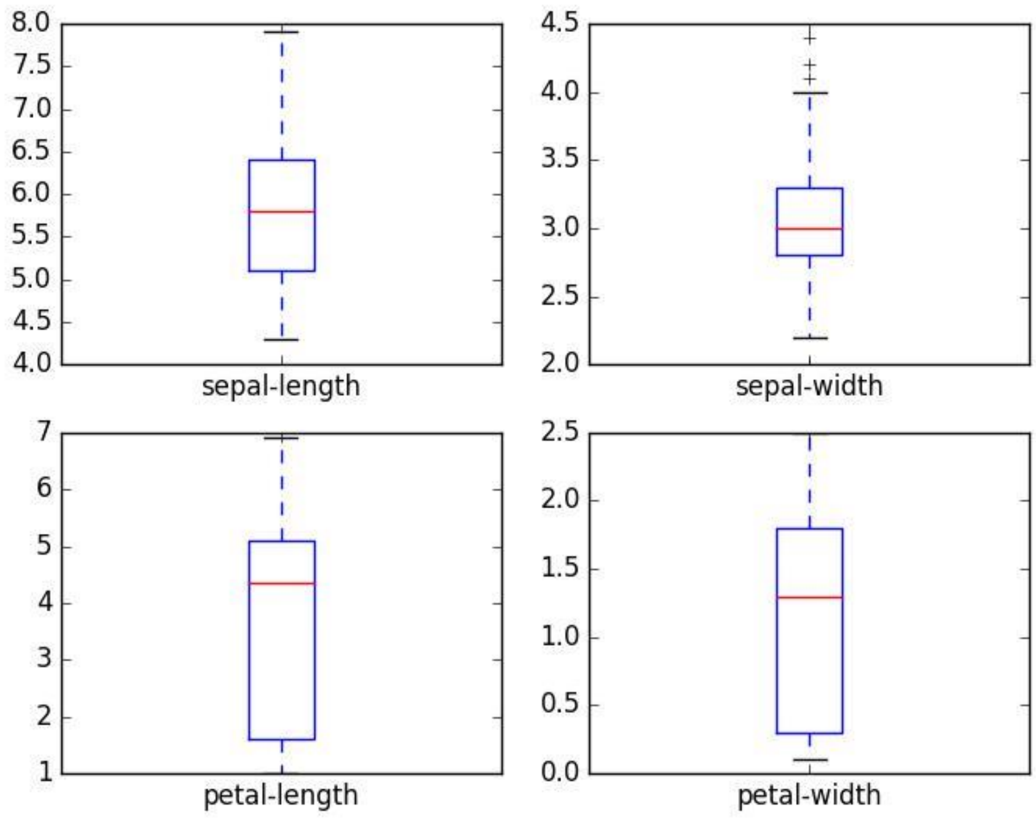
Univariate Plots

We start with some univariate plots, that is, plots of each individual variable.

Given that the input variables are numeric, we can create box and whisker plots of each.

```
/*********************************************************
# box and whisker plots
dataset.plot(kind='box', subplots=True, laWet=(2,2), sharex=False, sharey=False)
plt.show()
/************************************************************
```

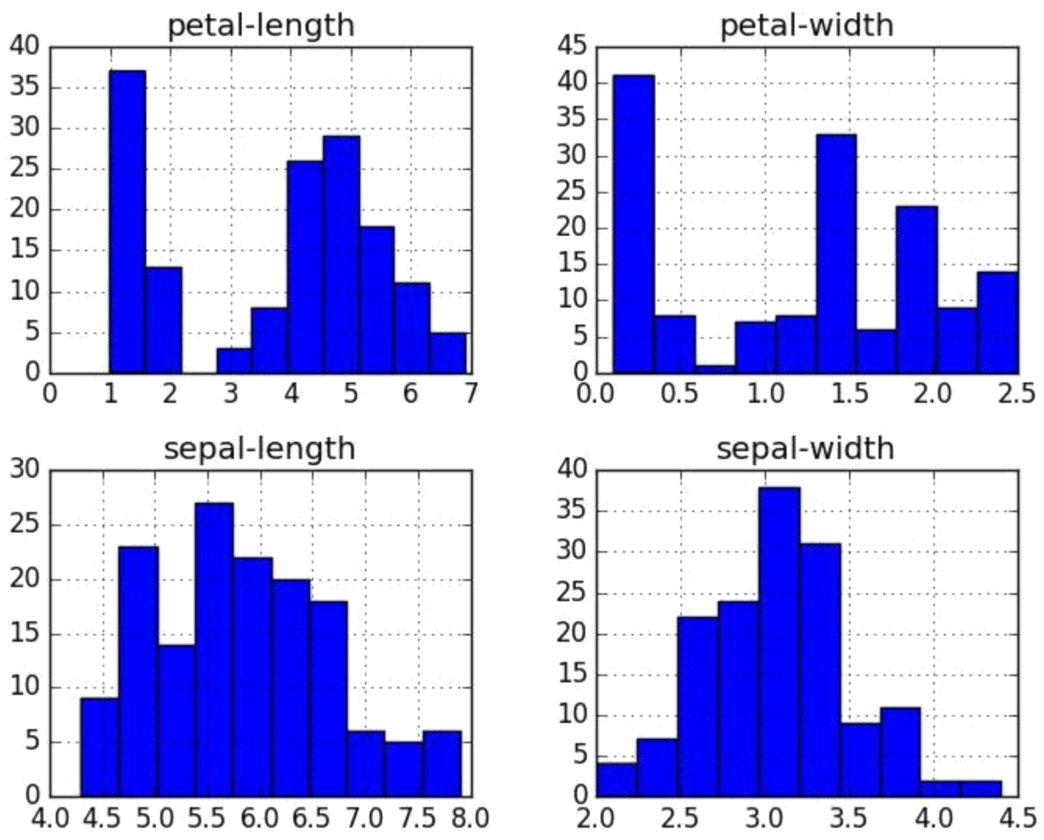
Box and Whisker plots



We can also create a histogram of each input variable to get an idea of the distribution.

```
/**************************************************
# histograms
dataset.hist()
plt.show()
/******************************************************
```

It looks like perhaps two of the input variables have a Gaussian distribution. This is useful to note as we can use algorithms that can exploit this assumption.



Histogram

Multivariate Plots

Using this we can look at the interactions between the variables.

Scatterplots (Pair plots) of all pairs of attributes. This is helpful to spot structured relationships between input variables.

```
/*******************************************

# 2D scatter plots

sns.set_style("whitegrid")

sns.FacetGrid(dataset, hue="class", height=4).map(plt.scatter, "sepal-length", "sepal-
width").add_legend()

plt.show()
```

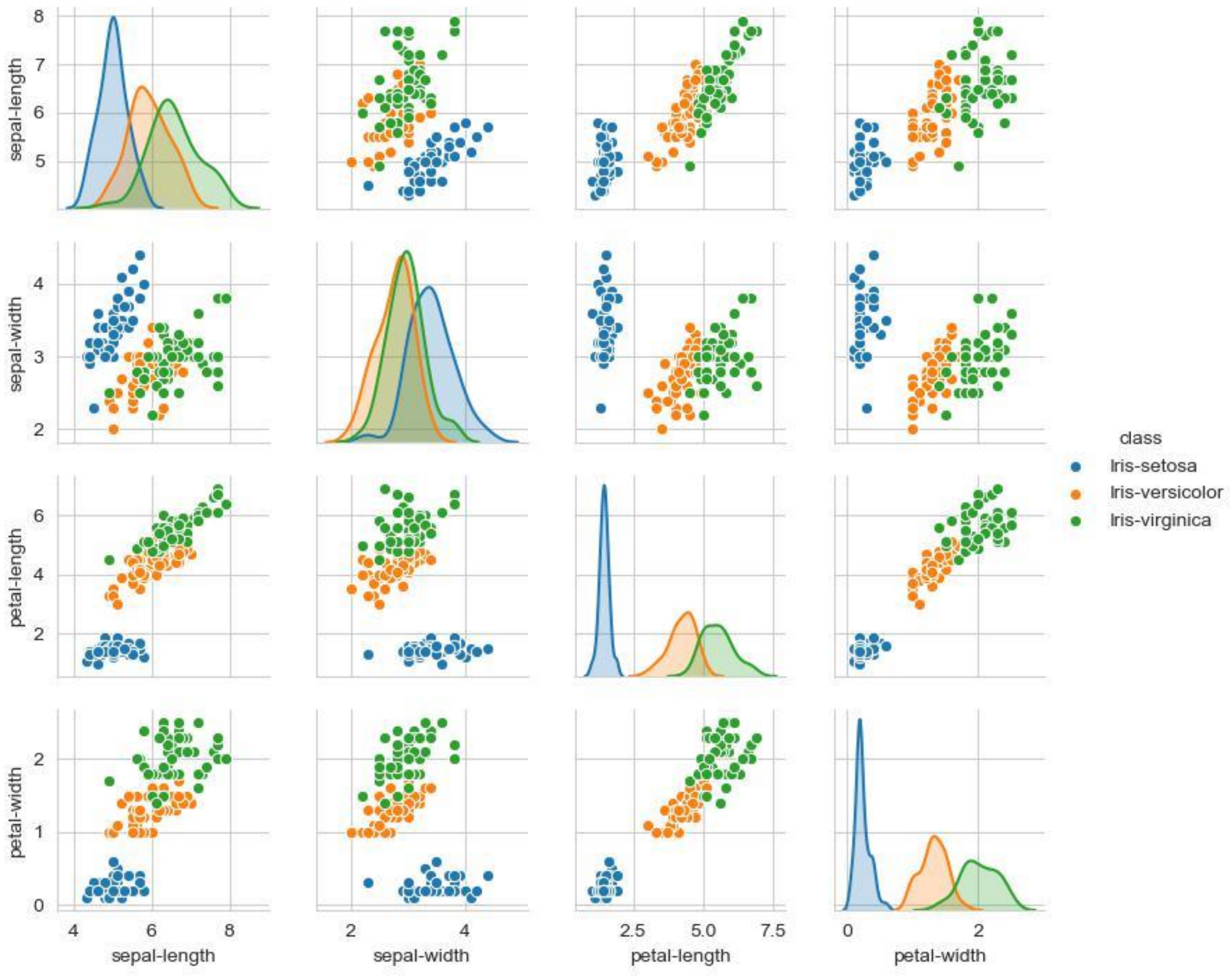
/**

/**

```
# pair-plots
sns.pairplot(dataset, hue="class", height=2)
plt.show();
```

/***

There is a diagonal grouping of some pairs of attributes, this suggests a high correlation and a predictable relationship.



Pair plot

Evaluating Some Algorithms

The next step is to create some models of the data and estimate their accuracy on unseen data.

Here is what we did to cover in this step:

- ✓ Separating out a validation dataset.
- ✓ Setting-up the test harness to use 10-fold cross validation.
- ✓ Building 5 different models to predict species from flower measurements
- ✓ Selecting the best model.

Creating a Validation Dataset

First, we checked whether the model we created is any good or not.

Later, we used statistical methods to estimate the accuracy of the models that we created on unseen data. We also wanted a more concrete estimate of the accuracy of the best model on unseen data by evaluating it on actual unseen data.

That is why, we held back some data that the algorithms didn't get to see and we used this data to get a second and independent idea of how accurate the best model might actually be.

We split the loaded dataset into two, 80% of which we used to train our models and 20% that we held back as a validation dataset.

```
/******************************
# Split-out validation dataset
array = dataset.values
X = array[:,0:4] Y =
array[:,4]
validation_size =
0.20 seed = 7
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X,
Y, test_size=validation_size, random_state=seed)
/************************************
```

Test Harness

We will use 10-fold cross validation to estimate accuracy.

This will split our dataset into 10 parts, train on 9 and test on 1 and repeat for all combinations of train-test splits.

```
/****************************************
# Test options and evaluation
metric seed = 7
scoring = 'accuracy'
/******************************************
```

The specific random seed does not matter, learn more about pseudorandom number generators here:

We used the **metric of '*accuracy*'** to evaluate models. **This is a ratio of the number of correctly predicted instances in divided by the total number of instances in the dataset multiplied by 100** to give a percentage (e.g. 95% accurate). We will be using the *scoring* variable when we run build and evaluate each model next.

Build Models

We don't know which algorithms would be good on this problem or what configurations to use. We get an idea from the plots that some of the classes are partially linearly separable in some dimensions, so we are expecting generally good results.

6 different algorithms used and evaluated:

- ✓ Logistic Regression (LR)
- ✓ Linear Discriminant Analysis (LDA)
- ✓ K-Nearest Neighbors (KNN).
- ✓ Classification and Regression Trees (CART).
- ✓ Gaussian Naive Bayes (NB).
- ✓ Support Vector Machines (SVM).

This is a good mixture of simple linear (LR and LDA), nonlinear (KNN, CART, NB and SVM) algorithms. We reset the random number seed before each run to ensure that the evaluation of each algorithm is performed using exactly the same data splits. It ensures the results are directly comparable.

```
/**********************************************************************

# Spot Check

Algorithms models = []

models.append(('LR', LogisticRegression()))

models.append(('LDA', LinearDiscriminantAnalysis()))

models.append(('KNN', KNeighborsClassifier()))

models.append(('CART', DecisionTreeClassifier()))

models.append(('NB', GaussianNB()))

models.append(('SVM', SVC()))

# evaluate each model in turn

results = []

names = []

for name, model in models:

    kfold = model_selection.KFold(n_splits=10, random_state=seed)

    cv_results = model_selection.cross_val_score(model, X_train, Y_train,
                 cv=kfold, scoring=scoring)

    results.append(cv_results)

    names.append(name)

    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())

    print(msg)

/***************************************************************************
```

Select Best Model

We now have 6 models and accuracy estimations for each. We need to compare the models to each other and select the most accurate.

Running the example above, we get the following raw results:

LR: 0.966667 (0.040825)

LDA: 0.975000 (0.038188)

KNN: 0.983333 (0.033333)

CART: 0.975000 (0.038188)

NB: 0.975000 (0.053359)

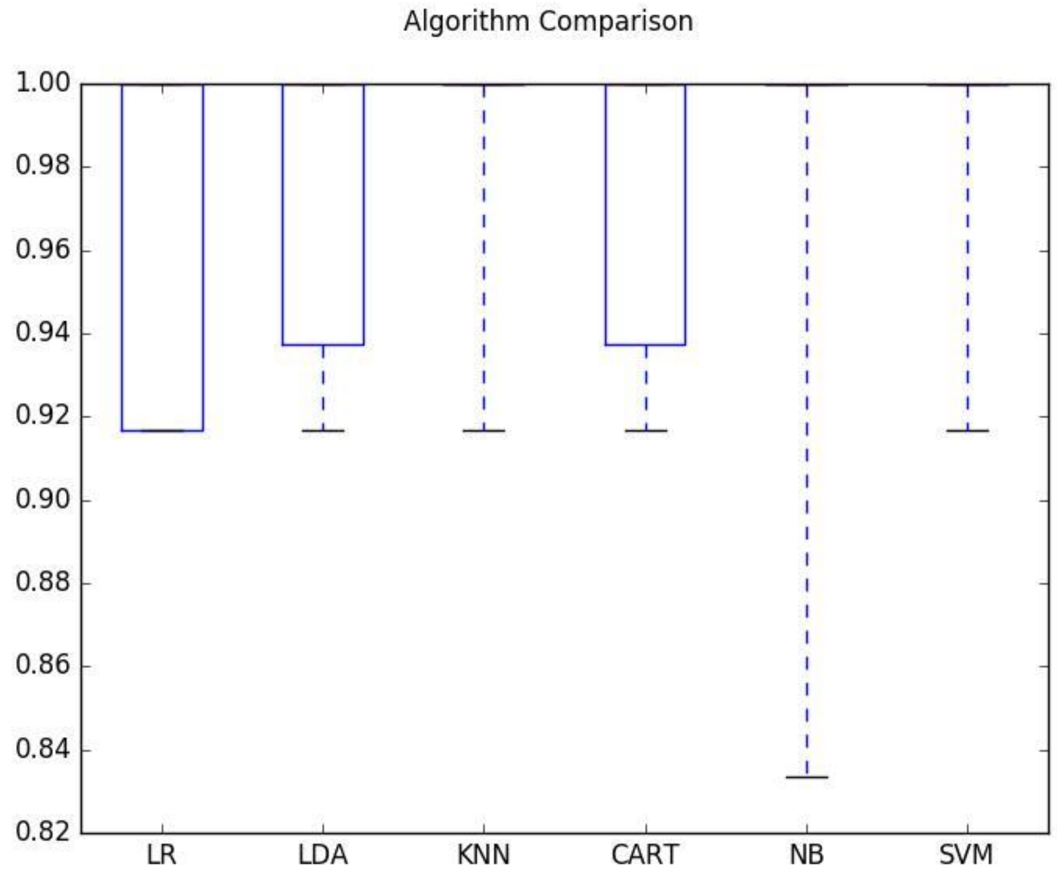
SVM: 0.981667 (0.025000)

It looks like KNN has the largest estimated accuracy score.

We can also create a plot of the model evaluation results and compare the spread and the mean accuracy of each model. There is a population of accuracy measures for each algorithm because each algorithm was evaluated 10 times (10-fold cross validation).

```
/**************************************************
# Compare Algorithms fig = plt.figure()
fig.suptitle('Algorithm Comparison') ax =
fig.add_subplot(111) plt.boxplot(results)
ax.set_xticklabels(names) plt.show()
/***********************************************
```

Note: the box and whisker plots are squashed at the top of the range, with many samples achieving 100% accuracy.



Predictions and Conclusions

The SVM algorithm was the most accurate model that we tested. Now we want to get an idea of the accuracy of the model on our validation set.

This gives us an independent final check on the accuracy of the best model. It is valuable to keep a validation set just in case We made a slip during training, such as overfitting to the training set or a data leak. Both will result in an overly optimistic result.

We can run the SVM model directly on the validation set and summarize the results as a final accuracy score, a confusion matrix and a classification report.

```
/***************************************************
# Make predictions on validation
dataset knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
predictions = knn.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
/*******************************************************
```

We can see that the accuracy is 0.99 or 99%. The confusion matrix provides an indication of the three errors made. Finally, the classification report provides a breakdown of each class by precision, recall, f1-score and support showing excellent results (granted the validation dataset was small).

```
 0.93

  [[ 7   0    0]
   [ 0  10    2]
   [ 0   0   11]]
```

	Precision	Recall	f1-Score	Support
Iris-setosa	1.00	1.00	1.00	7
Iris-versicolor	1.00	0.83	0.91	12
Iris-virginica	0.85	1.00	0.92	11
avg / total	0.94	0.93	0.93	30

References

https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7

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

https://seaborn.pydata.org/installing.html

https://machinelearningmastery.com/machine-learning-in-python-step-by-step/

https://python.org

****
*