Artificial Intelligence with Python

Classification cont.

Classification

In machine learning, classification means the prediction of the value of a categorical variable (having finitely many different values) based on training data.

In other words, the input data is to be classified or labeled using categories such as 0 or 1.

Algorithms for binary classification include

- logistic regression
- k-nearest neighbors ("knn")
- decision trees
- support vector machine (SVM)

Support vector machines

A Support Vector Machine (SVM) is a powerful supervised machine learning algorithm used for classification.

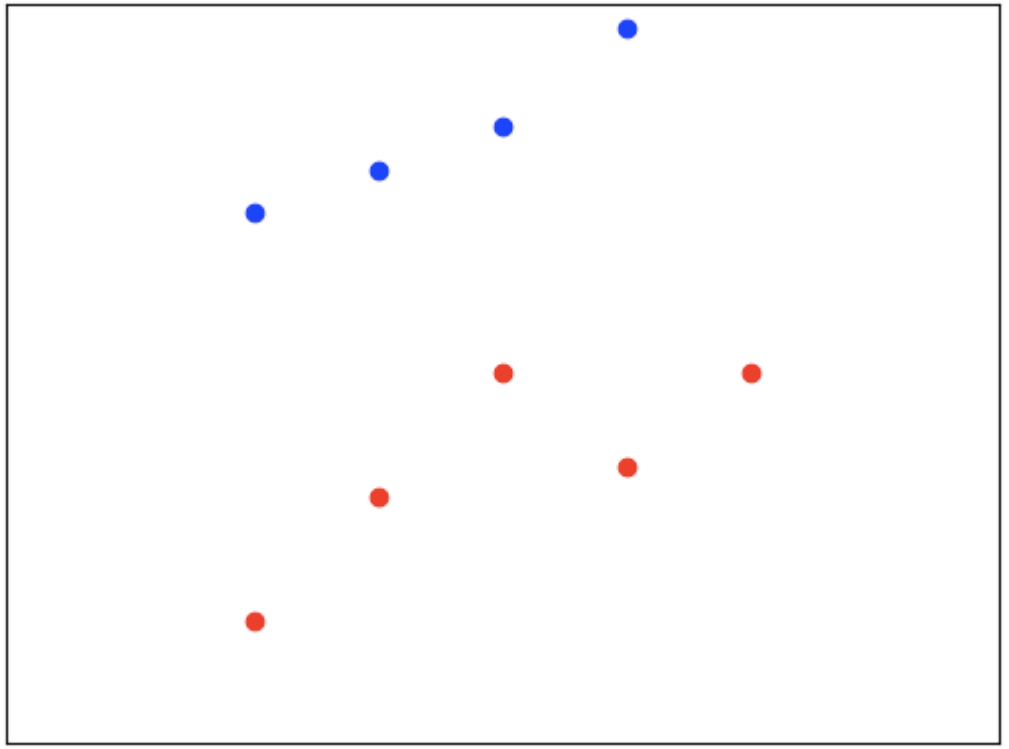
It's particularly well-suited for classification of complex datasets with a clear margin of separation between classes.

How SVM Works?

- SVM aims to find the optimal hyperplane that best separates data points belonging to different classes in the feature space.
- The hyperplane is the decision boundary that maximizes the margin between classes. It's defined by a set of support vectors, which are data points closest to the hyperplane.

Support vector machines

Suppose we are dealing with a binary classification problem with the following observations where red and blue dots represents the two different classes. We have two explanatory variables here.

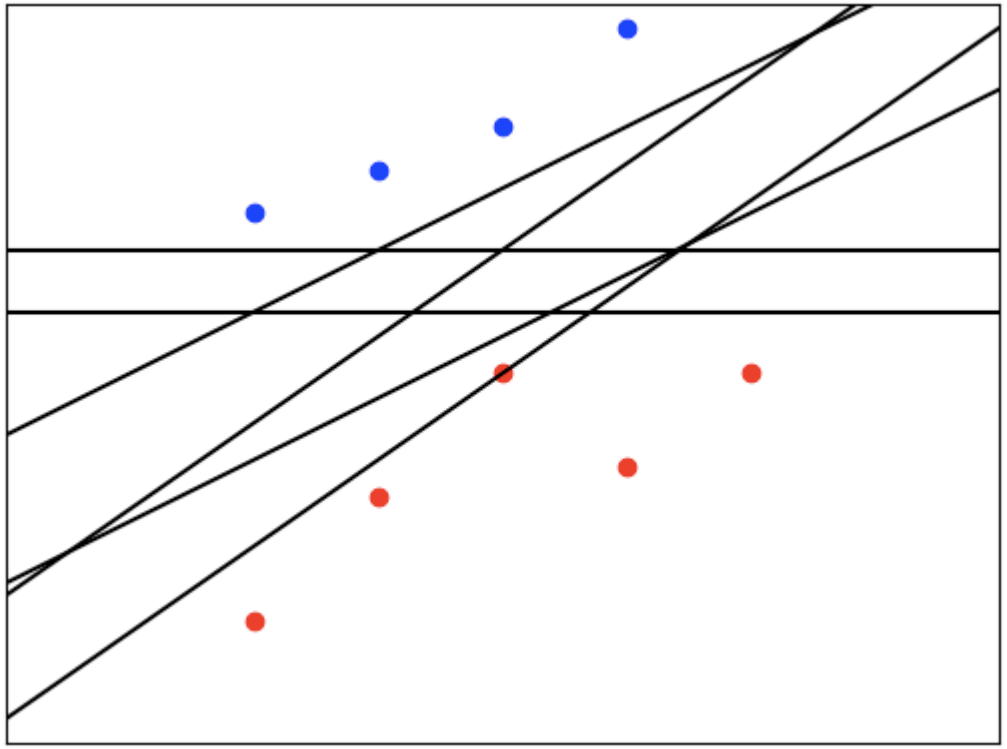


Margin in SVM

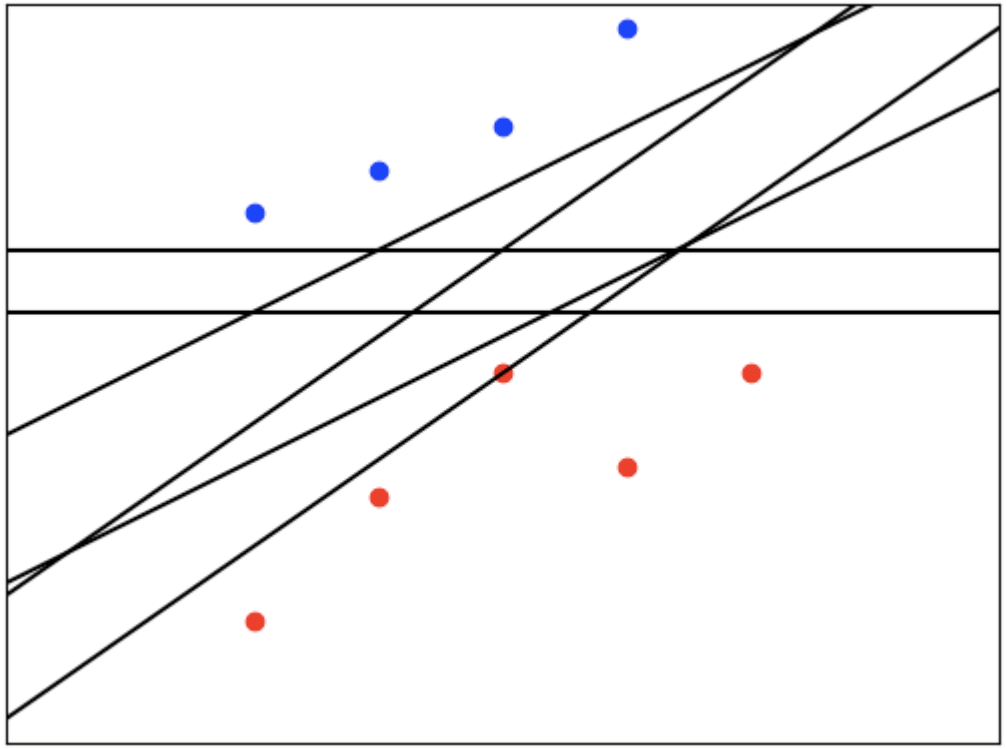
The margin is the distance between the hyperplane and the closest data points from each class (support vectors).

SVM aims to maximize this margin, as it leads to better generalization and robustness of the classifier.

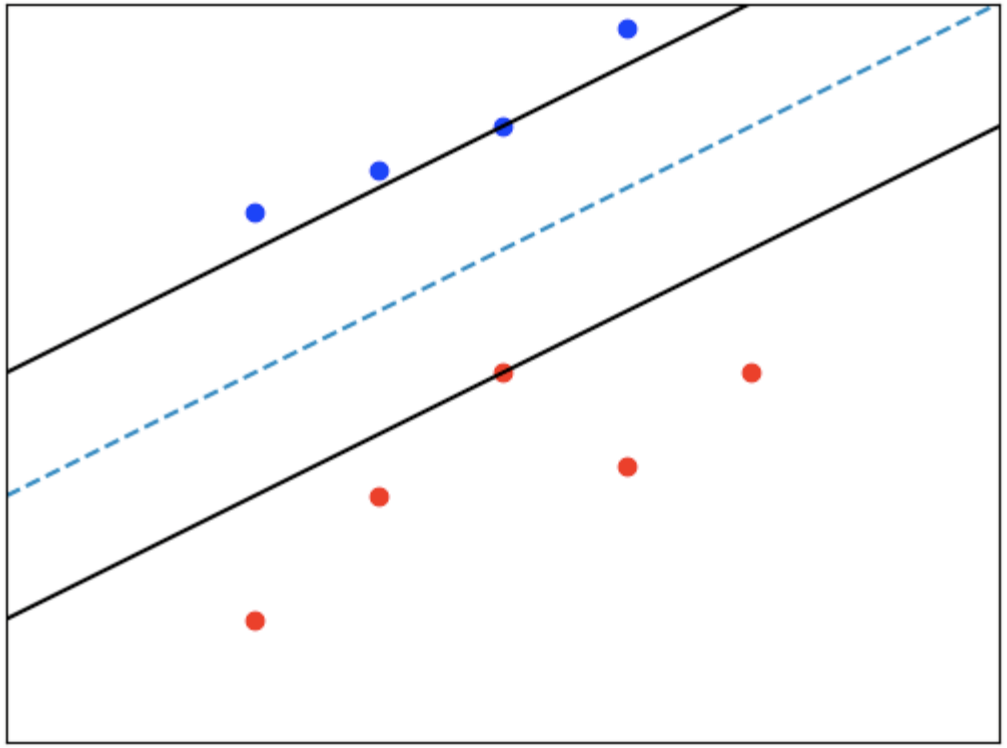
We would like to find a way to separate the classes by a decision boundary. The simplest way here would be to draw straight lines such as below.



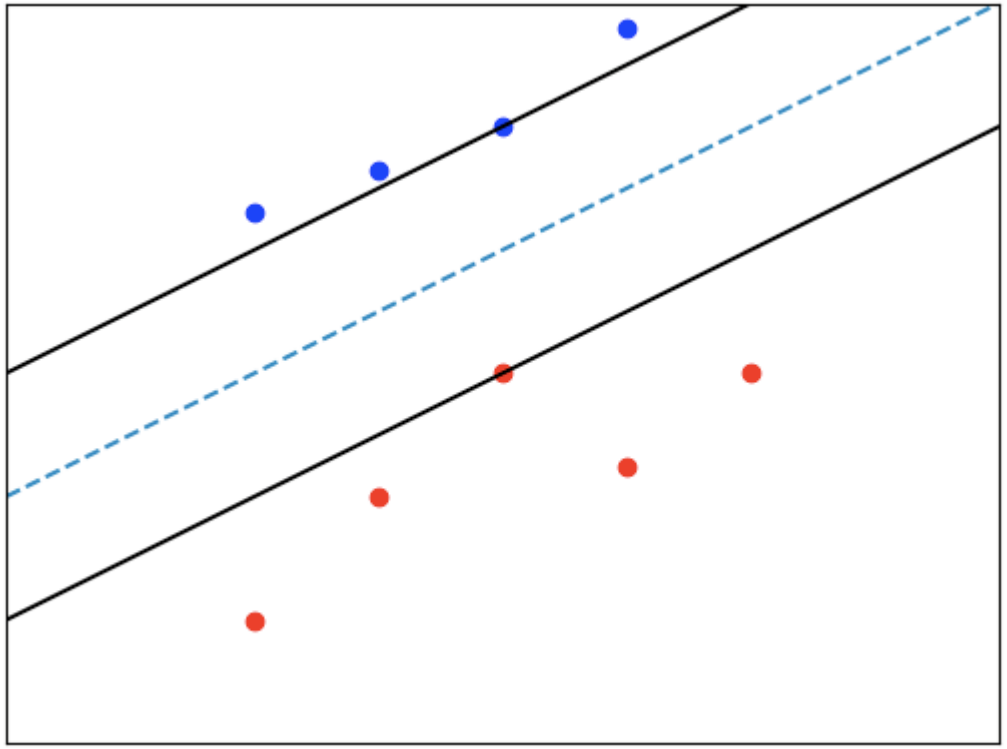
Note that the lines may be horizontal (slope zero at different heights) or they may be increasing with some slope and some y-intercept. **The key question here is which line to choose?**



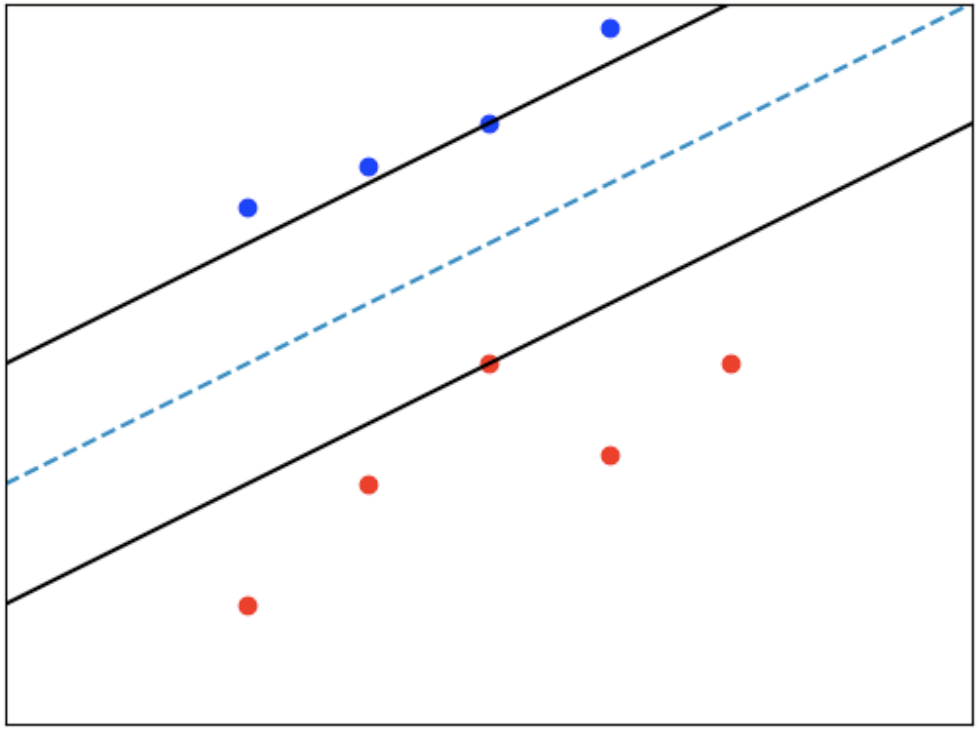
The optimal choice is the one that maximizes the separation between the classes because that provides the largest margin of error for misclassification.
In this particular case it would look like this:



The blue dashed line and black solid lines form "a street" and the median line (blue dashed line) is the one that maximizes the separation. We also say that it provides the largest margin between the samples. In the prediction step, points above the median line would be classified as blue and the points below as red. That acts as the decision boundary.



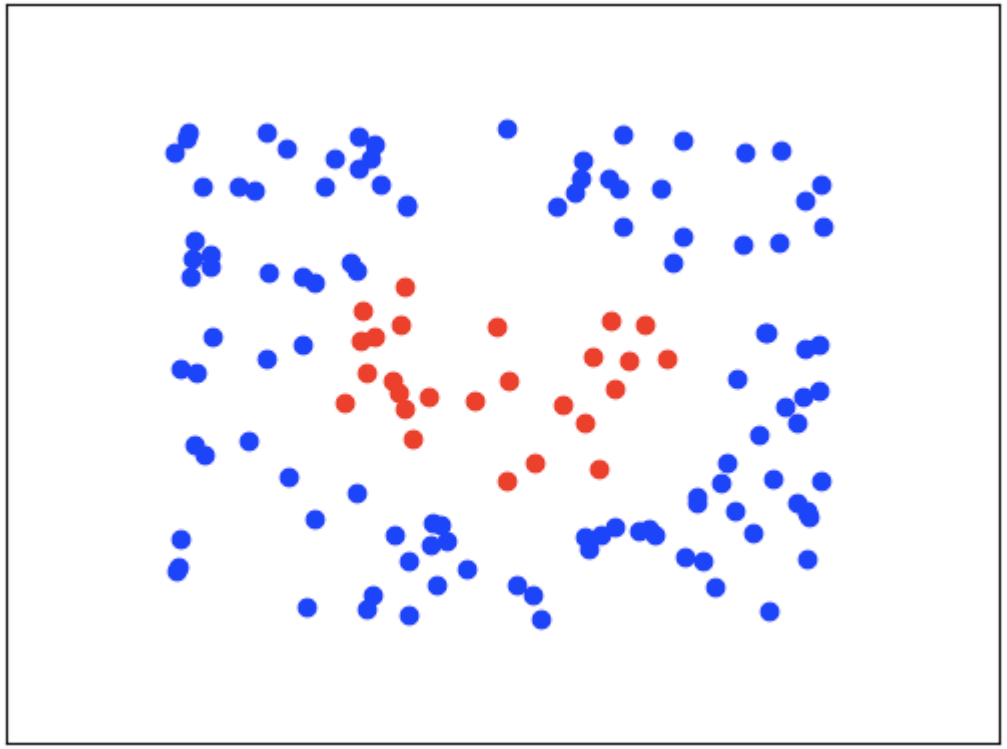
- In the picture the single blue and red dots touching the solid black lines are called the support vectors.



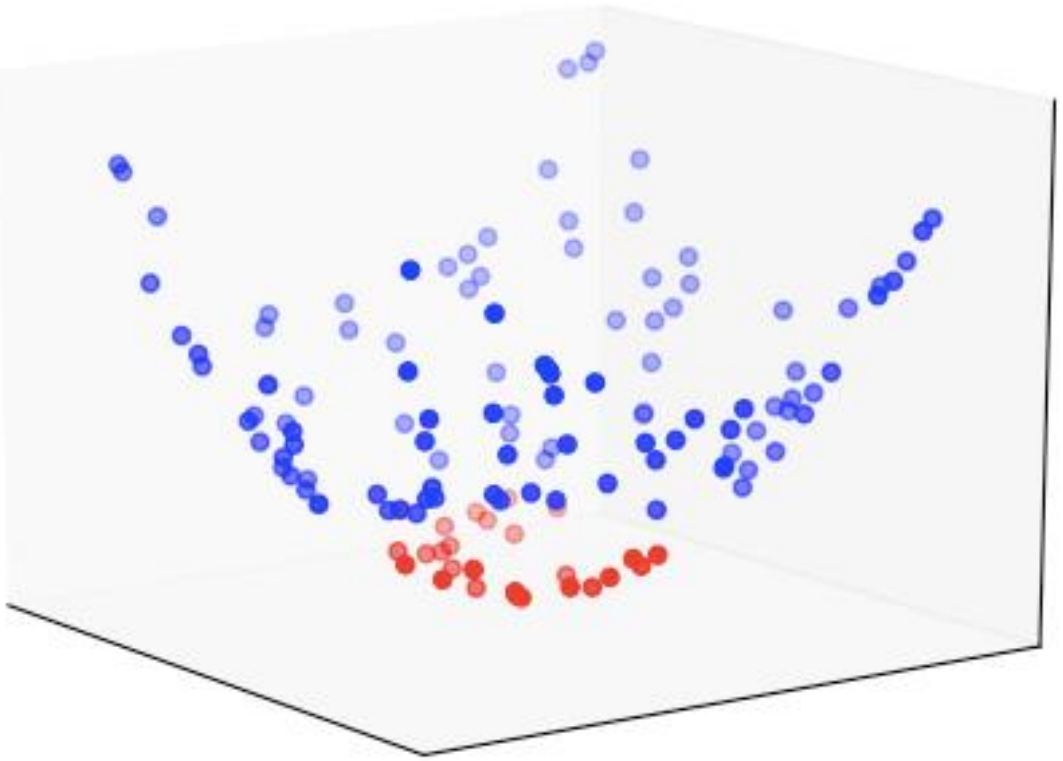
The same idea works in higher dimensions too.

If we have three explanatory variables then the scatter plot of the data points is three dimensional and we look for a plane separating the two classes.

This idea works well in cases where classes can be separated by a straight line (or plane). But what about if the data points were not linearly separable? For example, we might have the following situation:



There is no way to separate red and blue dots by a straight line. The trick here is to transform the data points to three dimensional space. If we use, for example, the mapping z = x^2 + y^2 then we get the following scatter plot



Now red and blue dots can be separated by a plane in three dimensions.

That's why the separation boundary is called the hyperplane since it works in any dimensions.

There are many possibilities for the nonlinear transformation such as polynomial, exponential (radial basis function, RBF), sigmoid and so on.

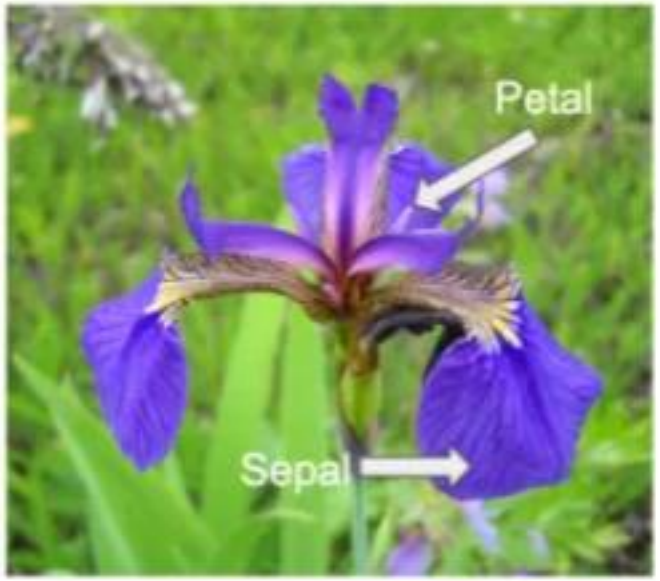
Kernel Function

1. SVM can efficiently handle both linearly separable and non-linearly separable datasets through the use of kernel functions.
2. Kernel functions transform the input features into higher-dimensional space where the classes become more separable.
3. Common kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid.

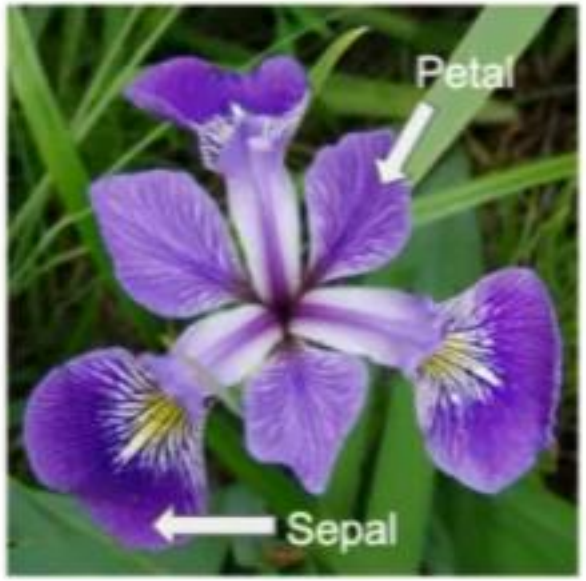
To show SVM in action we apply it to classify iris flowers dataset. In the svm module, sklearn provides SVC() which stands for support vector classifier.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

from sklearn.metrics import classification_report, confusion_matrix
```

Iris setosa



Iris versicolor



Iris virginica

```
df2 = pd.read_csv("iris.csv")
print(df2.head())
X = df2.drop('species', axis=1)
y = df2['species’]
```

Split data as before

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state=20)
```

We first use the linear kernel in SVC() when fitting the model and making predictions.

```
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)
```

Model evaluation shows nearly perfect performance

```
print(confusion_matrix(y_test,y_pred))

print(classification_report(y_test,y_pred))
```

Nevertheless, the data remains not linearly separable so we shall try other kernel functions too.
Second degree polynomial shows better performance

```
svclassifier = SVC(kernel='poly', degree=2)

svclassifier.fit(X_train, y_train)

y_pred = svclassifier.predict(X_test)

print(confusion_matrix(y_test,y_pred))

print(classification_report(y_test,y_pred))
```

Using radial basis function (Gaussian exponential function) leads to similar accuracy

```
svclassifier = SVC(kernel='rbf')

svclassifier.fit(X_train, y_train)

y_pred = svclassifier.predict(X_test)

print(confusion_matrix(y_test,y_pred))

print(classification_report(y_test,y_pred))
```

Feature scaling

Feature scaling

When dealing with multiple explanatory variables it often makes a difference in machine learning models how the variables are scaled. In other words, some variable may contain large numbers and another only small values. Or the numbers might be of opposite sign.

Algorithms based on distance measurements (e.g. KNN) are prone to perform better if all variables are in same scale.

There are two common methods to rescale values; normalization and standardization.

Normalization means that we subtract the minimum value from all numbers and divide the result by the difference between smallest and largest value. As formula

$$X_{norm} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

This implies that all numbers are transformed to between 0 and 1. This happens because for Xmin numerator becomes zero and for Xmax it becomes the same as denominator. This scaling is also called the min-max scaling.

$$X_{std} = \frac{X - \overline{X}}{std(X)}$$

Standardization means that we subtract the mean of numbers and divide by the standard deviation. The formula is thus

The resulting numbers will have mean of zero and standard deviation of unit. It means that the transformed data will be distributed like standard normal distribution (Gaussian distribution). Hence it is usually used for variables which obey a Gaussian distribution.

It is also worthwhile to note that standardization does not put bounds on new values whereas normalization does.

Note also that both scaling techniques lose information about the unit of numbers.

sklearn provides both of these two scaling techniques in the preprocessing module as **MinMaxScaler and StandardScaler.**

Let us work out a practical understanding for feature scaling. As numerical data we use the students' heights and weights that we have met before.

```
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler, StandardScaler

import pandas as pd

import numpy as np

df = pd.read_csv('weight-height.csv',skiprows=0,delimiter=",")

X = 2.54*df[['Height']]
```

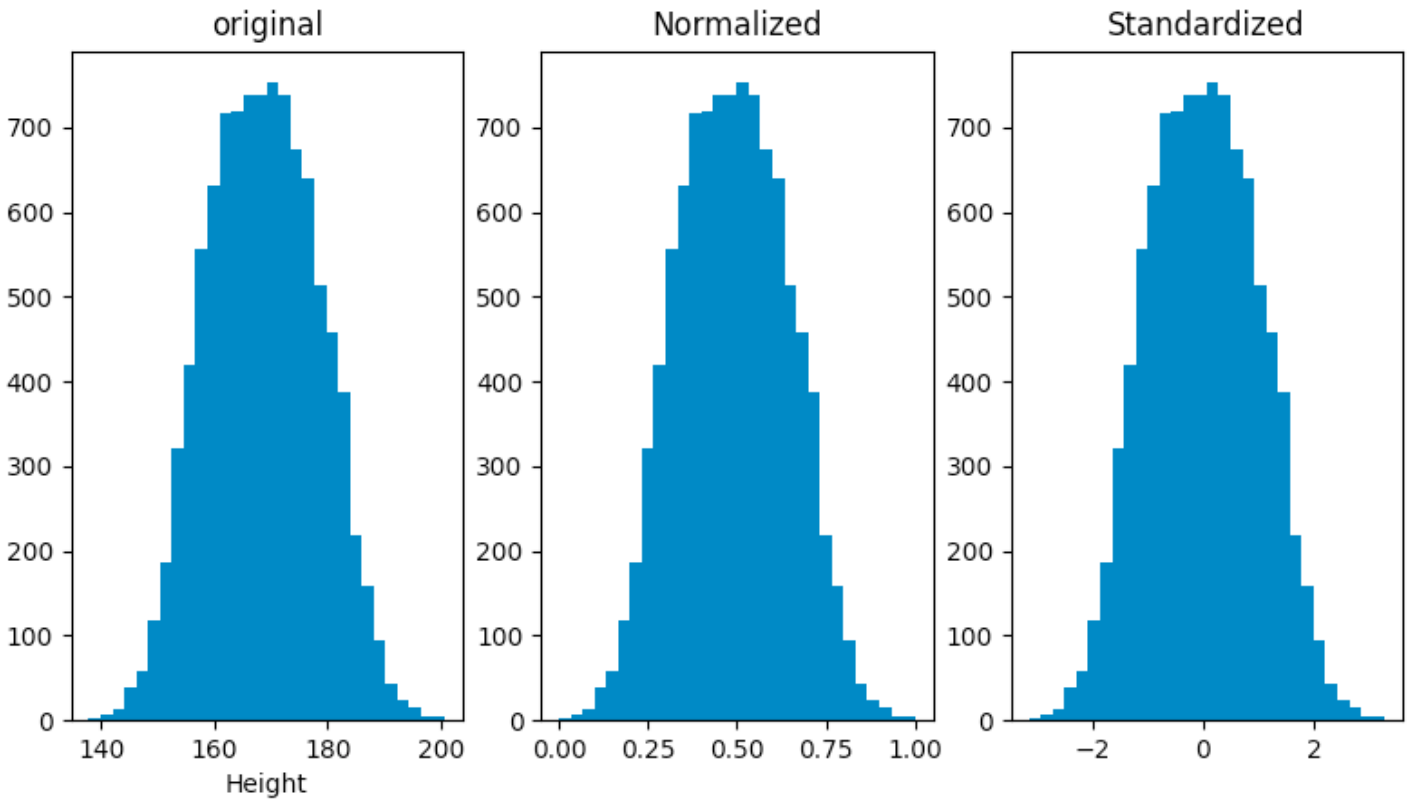
Data is scaled using sklearn as follows

```
X_mm = MinMaxScaler().fit_transform(X)
X_std = StandardScaler().fit_transform(X)
```

We plot the histograms of original and scaled data sets

```
X = np.array(X)

X_mm = np.array(X_mm)

X_std = np.array(X_std)
```

```
plt.subplot(1,3,1)

plt.hist(X,30)

plt.xlabel("Height")

plt.title("original")

plt.subplot(1,3,2)

plt.hist(X_mm,30)

plt.title("Normalized")

plt.subplot(1,3,3)

plt.hist(X_std,30)

plt.title("Standardized")

plt.show()
```


original
Normalized
Standardized
700
600
500
400
300
200
100
0
140
160
180
200
Height
0.00
0.25
0.50
0.75
1.00
−2
0
2

We observe that the original data of heights was already distributed quite normally around the mean. Therefore both scalings retain the same form of histograms.

We could have done the scaling using the low-level formulas as well and checked that they give the same results.

```
X_mm2 = (X-np.min(X))/(np.max(X)-np.min(X))

print("diff=",np.max(np.abs(X_mm-X_mm2)))


X_std2 = (X-np.mean(X))/np.std(X)

print("diff2=",np.max(np.abs(X_std-X_std2)))
```

Effect of feature scaling in machine learning

Here we investigate how feature scaling affects the performance of a machine learning model.

As an example, we use the k-nearest neighbors algorithm for a regression problem. The algorithm works similarly as for classification problems and looks for nearest neighbors of a test point and predicts it's value to be the average of neighbors' values. However, this algorithm does not (even attempt to) find the regression line but only predict target variable.

We continue to consider the university admissions data.

```
import matplotlib.pyplot as plt
from sklearn import neighbors
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import pandas as pd
import numpy as np
df = pd.read_csv('Admission_Predict.csv',skiprows=0,delimiter=",")
print(df)
```

We pick the CPGA and GRE scores as our feature variables for this demonstration because their values are in different scales. The goal is to predict the chance of admit.

```
X = df[["CGPA",'GRE Score']]

y = df[["Chance of Admit "]]

print(X)

print(y)
```

Split the dataset into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=30)
```

Next, we scale the training and testing data using both scaling methods. Note that scaling applies only to the feature variables and not to the target variable y.

```
X_train_norm = MinMaxScaler().fit_transform(X_train)
X_test_norm = MinMaxScaler().fit_transform(X_test)
X_train_std = StandardScaler().fit_transform(X_train)
X_test_std = StandardScaler().fit_transform(X_test)
print(X_train_norm)
print(X_test_norm)
print(X_train_std)
print(X_test_std)
```

Using the three different datasets for training we fit the model and compute the R2 value (without predicting on the test set ourselves).

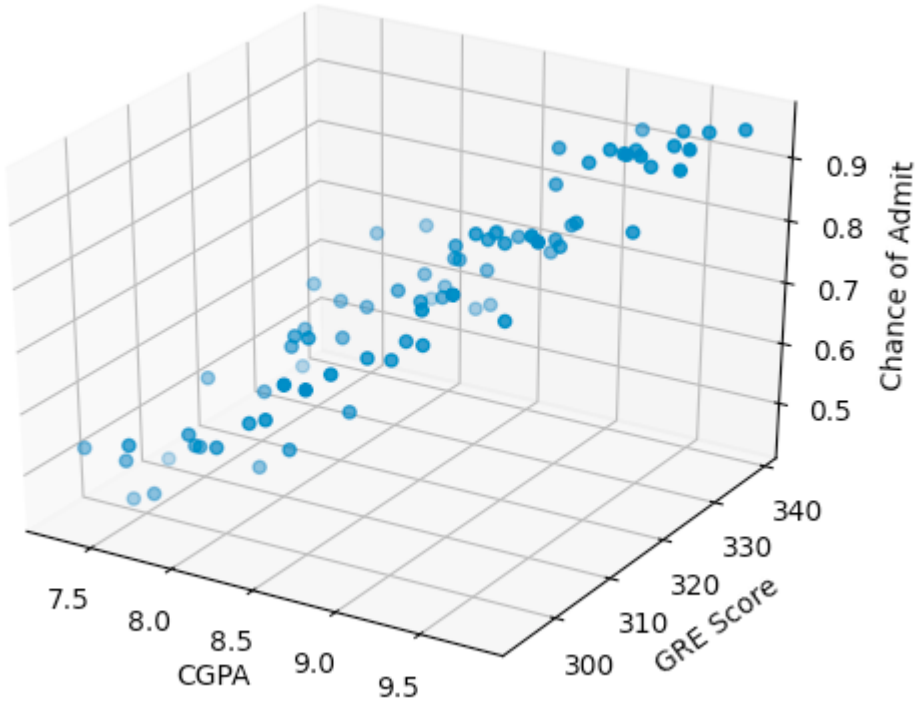
```
lm = neighbors.KNeighborsRegressor(n_neighbors=5)
```

```
lm.fit(X_train, y_train)
predictions = lm.predict(X_test)
print("R2 =",lm.score(X_test,y_test))

lm.fit(X_train_norm, y_train)
predictions2 = lm.predict(X_test_norm)
print("R2 (norm) =",lm.score(X_test_norm,y_test))

lm.fit(X_train_std, y_train)
predictions3 = lm.predict(X_test_std)
print("R2 (std) =",lm.score(X_test_std,y_test))
```

The R2 value is 0.75 for unscaled data, 0.75 for normalized data and 0.81 for standardized data. So we got slightly better performance by using standardized features.

For completeness, we stress that predictions are now not in any one line (we are not doing linear regression) but spread like this:

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(np.array(X_test[['CGPA']]),np.array(X_test[['GRE Score']]),predictions)
ax.set_xlabel('CGPA')
ax.set_ylabel('GRE Score')
ax.set_zlabel('Chance of Admit')
plt.show()
```


0.9
0.8
0.7
0.6
0.5
Chance of Admit
340
330
320
310
300
GRE Score
7.5
8.0
8.5
9.0
9.5
CGPA

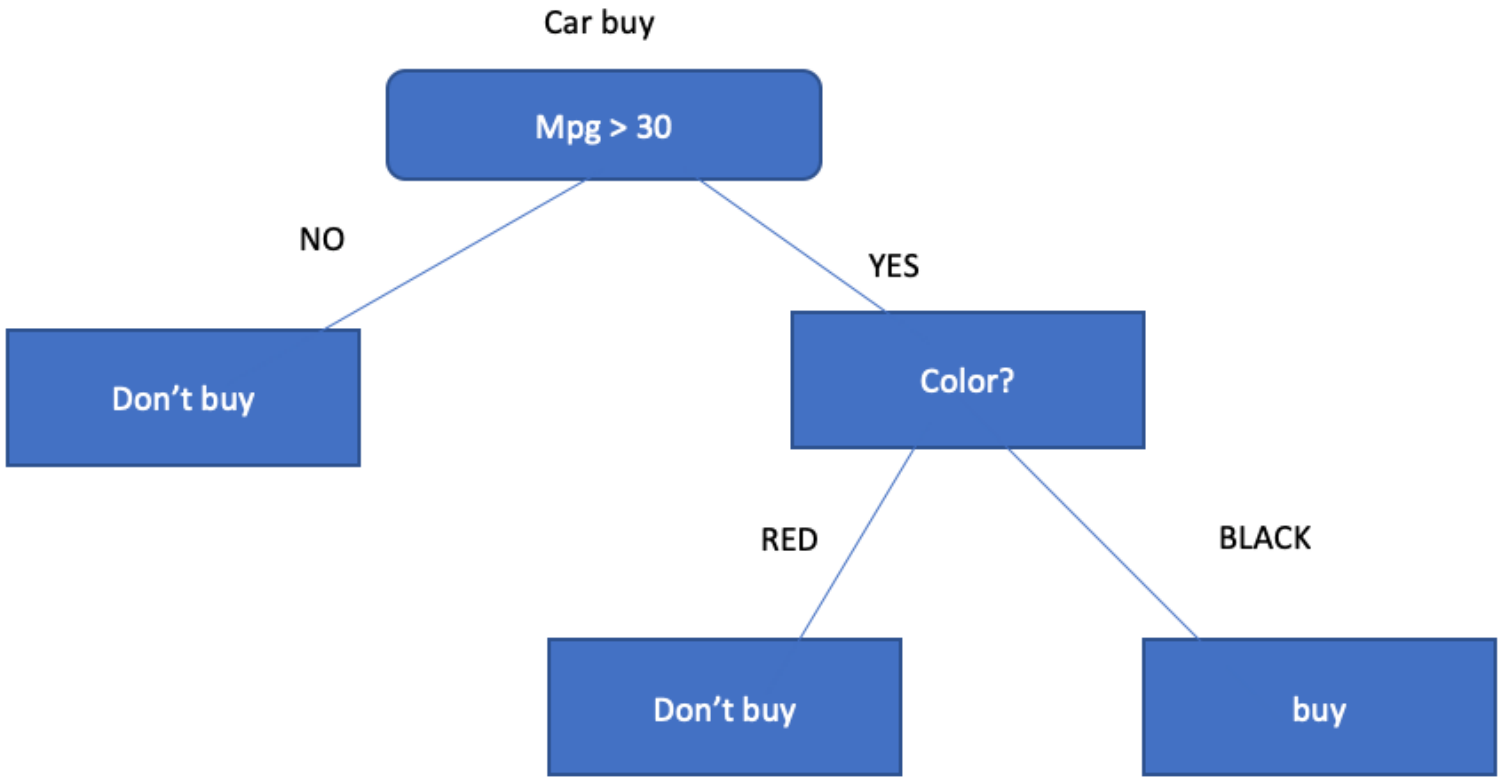
Decision trees

Decision trees are most often applied to classification problems although they can be used for regression too with appropriate modifications.

Roughly speaking, decision trees are most suitable for complicated problems where the data contains complex relationships between variables. Ensembles of decision trees lead to random forests which show their power for problems with large datasets.

A decision tree is a tree shaped diagram consisting of nodes and branches to make a decision. Leaves are at the very bottom of the tree. They have no branches and they represent a final decision.

For example, we could make decisions when buying a car based on mileage and color etc.



Starting from original full dataset we make a sequence of decisions and split the dataset into smaller parts. Eventually, at leaf nodes, there is nothing to split and we are done. This process leads us to some key terms in decision trees:

entropy = amount of randomness (or unpredicatability) in a dataset

information gain = decrease in entropy after splitting a dataset

Gini index = value between 0 and 1, where 0 means that all elements belong to same class and 1 means that all elements are randomly distributed across various classes. So larger values indicate more randomness or impurity.

At the very beginning, entropy is high in the original dataset. After several splits the entropy decreases and we gain more information.

There is actually a mathematical formula to compute the entropy of a dataset. It reads

$$E = -\sum P_i \log_2 P_i$$

where Pi is the probability of sample taking on i th value. For example, consider the following datasets consisting of red, blue, green, and yellow marbles. Both sets contain 16 marbles.

	set 1	set 2
red	4	1
blue	4	1
green	4	7
yellow	4	7

Set 1 has the entropy score of

$$E = -(\frac{1}{4}\log_2\frac{1}{4} + \frac{1}{4}\log_2\frac{1}{4} + \frac{1}{4}\log_2\frac{1}{4} + \frac{1}{4}\log_2\frac{1}{4}) = 2$$

while set 2 has the entropy

$$E = -(\frac{1}{16}\log_2\frac{1}{16} + \frac{1}{16}\log_2\frac{1}{16} + \frac{7}{16}\log_2\frac{7}{16} + \frac{7}{16}\log_2\frac{7}{16}) = 1.544$$

So set 1 has higher entropy.

For Gini index the formula is

$$G = 1 - \sum P_i^2$$

In the above example, set 1 has Gini index of 1-4*(1/16)= 0.75 and set 2 has 0.61. So, again, Set 1 contains more impurities.

These measures of impurity and randomness can be used to construct a decision tree in an optimal manner. More precisely, they help us pick the features which decrease entropy the most first.

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.model_selection import train_test_split
import pandas as pd
import graphviz

df = pd.read_csv("data_banknote_authentication.csv")
print(df.head())
```

```
#We use all but last column as features and the last column "class" as our target variable


X = df.drop('class', axis=1)
y = df['class']
print(X)
print(y)


#Split data using 80/20 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,random_state=11)
```

Set up decision tree classifier with default parameters, train and predict

```
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

Evaluation is based on confusion matrix and metrics

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

We received 99% accuracy and high precision and recall scores too.

THANKS