

Candidate Elimination Algorithm

The candidate elimination algorithm makes use of hypothesis space H and a set of examples E to incrementally build a version space. Each example in E is added one by one. An example may shrink the version space by eliminating the hypotheses that are inconsistent with the example. With each new example, the algorithm updates the general and specific boundary.

Both, positive as well as negative examples are considered. While positive examples are used for generalizing the specification, negative examples on the other hand are specified from generalized form.

In the candidate elimination algorithm, we make machines learn by using two sets – General Hypothesis and Specific Hypothesis. Where,

General Hypothesis does not specify any features to make the machine learn. We just put n number of '?' in this set where n is the number of attributes. Thus,

$$G = \{ '?', '?', '?', '?', \dots \}$$

Specific Hypothesis is the set of features that help machines learn. Initially, $S = \{ \text{NULL}, \text{NULL}, \text{NULL}, \dots \}$ where Number of NULL depends on the number of attributes.

Version Space sits between general hypothesis and specific hypothesis. It is a subset of H . version space contains only those hypotheses from H that are consistent with an observed sequence of training examples.

Algorithm

Step 1: Load dataset.

Step 2: Initialize General Hypothesis and Specific Hypothesis.

Step 3: For each training example.

Step 4: If example is positive example

```
if attribute_value == hypothesis_value:
    Do nothing
else:
    replace attribute value with '?' to generalize it
```


Make generalize hypothesis more specific.

Sky	Temperature	Humidity	Wind	Water	Forest	Output
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

Initially: $G = [[?, ?, ?, ?, ?], [?, ?, ?, ?, ?], [?, ?, ?, ?, ?], [?, ?, ?, ?, ?], [?, ?, ?, ?, ?], [?, ?, ?, ?, ?]]$

For example 1: <'sunny','warm','normal','strong','warm ','same'> Since the output is positive, general hypothesis remains the same but we will make changes in our specific hypothesis.

```
S1 = ['sunny', 'warm', 'normal', 'strong', 'warm ', 'same']
```

```
S2 = ['sunny', 'warm', '?', 'strong', 'warm', 'same']
```

G3 = [['sunny', '?', '?', '?', '?', ?], [?, 'warm', '?', '?', '?', ?], [?, '?', '?', '?', '?', ?], [?, '?', '?', '?', '?', ?],
[?, '?', '?', '?', '?', 'same']]

For example 4: <'sunny','warm','high','strong','cool','change'> and positive output.

```
S4 = ['sunny', 'warm', '?', 'strong', '?', '?']
```

At last, by synchronizing the G4 and S4 algorithm produce the output.

Output:

```
G = [['sunny', '?', '?', '?', '?', '?'], [?, 'warm', '?', '?', '?', ?]]
```

```
S = ['sunny', 'warm', '?', 'strong', '?', '?']
```


Example: Consider the dataset given below and apply candidate elimination algorithm on it.

Article	Crime	Academic	Local	Music	Reads
a ₁	true	false	false	true	true
a ₂	true	false	false	false	true
a ₃	false	true	false	false	false
a ₄	false	false	true	false	false
a ₅	true	true	false	false	true

Technique 1:

Let $G_0 = \{true\}$ if the user reads everything and $S_0 = \{false\}$ when the user reads nothing.

After considering the first example, a_1 , $G_1 = \{true\}$ and

$S_1 = \{crime \wedge \neg academic \wedge \neg local \wedge music\}$.

Now, the most general hypothesis states that the user reads everything, and the most specific hypothesis says that the user only reads articles exactly like this one.

After considering the second example, we get, $G_2 = \{true\}$ and

$S_2 = \{crime \wedge \neg academic \wedge \neg local\}$.

This is because the second example adds information that music cannot be relevant.

After considering the third example,

$G_3 = \{crime, \neg academic\}$ and $S_3 = S_2$.

Now there are two most general hypotheses; the first is that the user reads anything about crime, and the second is that the user reads anything non-academic.

After considering the fourth example,

$G_4 = \{crime, \neg academic \wedge \neg local\}$ and $S_4 = S_3$.

Once all five examples are analyzed, we conclude that

$G_5 = \{crime\}$, $S_5 = \{crime \wedge \neg local\}$.

Technique 2:

Initially: $G = [[?, ?, ?, ?], [?, ?, ?, ?], [?, ?, ?, ?], [?, ?, ?, ?]]$

$S = [Null, Null, Null, Null]$

For instance 1: $\langle 'true', 'false', 'false', 'true' \rangle$ and positive output.

$G_1 = G$

$S_1 = ['true', 'false', 'false', 'true']$

For instance 2: $\langle 'true', 'false', 'false', 'false' \rangle$ and positive output.

$G_2 = G$

$S_2 = ['true', 'false', 'false', '?']$

For instance 3: $\langle 'false', 'true', 'false', 'false' \rangle$ and negative output.

$G_3 = [['true', ?, ?, ?], [?, 'false', ?, ?]]$

$S_3 = ['true', 'false', 'false', '?']$

For instance 4: $\langle 'false', 'false', 'true', 'false' \rangle$ and negative output.

$G_4 = [['true', ?, ?, ?], [?, ?, 'false', ?]]$

$S_4 = ['true', 'false', 'false', '?']$

For instance 5: $\langle 'true', '?', 'false', '?' \rangle$ and positive output.

$G_5 = [['true', ?, ?, ?]]$

$S_5 = ['true', '?', 'false', '?']$

Copyright © 2023 by McGraw Hill Education (India) Private Limited

Example: Consider the dataset given below and find the value of H and S.

Example	Citations	Size	InLibrary	Price	Editions	Buy
1	Some	Small	No	Affordable	One	No
2	Many	Big	No	Expensive	Many	Yes
3	Many	Medium	No	Expensive	Few	Yes
4	Many	Small	No	Affordable	Many	Yes

Technique 1: Start with a negative example.

Initially: $G = [[?, ?, ?, ?, ?], [?, ?, ?, ?, ?], [?, ?, ?, ?, ?], [?, ?, ?, ?, ?], [?, ?, ?, ?, ?]]$

$S = [Null, Null, Null, Null, Null]$

For instance 1: <'some', 'small', 'no', 'affordable', 'one'> negative output.

$G1 = [['many', ?, ?, ?, ?], [?, b, ?, ?, ?], [?, 'many', ?, ?, ?], [?, ?, 'expensive', ?, ?], [?, ?, ?, ?, 'many'], [?, ?, ?, ?, 'few']]$

$S1 = S$

For instance 2: positive output.

$G1 = [['many', ?, ?, ?, ?], [?, 'big', ?, ?, ?], [?, ?, 'expensive', ?, ?], [?, ?, ?, ?, 'many']]$

$S1 = ['many', 'big', 'no', 'expensive', 'many']$

For instance 3: positive output.(retain only those applicable in G)

$G1 = [['many', ?, ?, ?, ?], [?, ?, 'expensive', ?, ?]]$

$S1 = ['many', ?, 'no', 'expensive', ?]$

For instance 4: positive output.(retain only those applicable in G)

$G1 = [['many', ?, ?, ?, ?]]$

$S1 = ['many', ?, 'no', ?, ?]$

Technique 2: Reorder examples to start with a positive example.

Initially:

$S = [Null, Null, Null, Null, Null]$

$G = [[?, ?, ?, ?, ?], [?, ?, ?, ?, ?], [?, ?, ?, ?, ?], [?, ?, ?, ?, ?], [?, ?, ?, ?, ?]]$

Instance 1

$S = ['many', 'big', 'no', 'affordable', 'one']$

$G = [[?, ?, ?, ?, ?], [?, ?, ?, ?, ?], [?, ?, ?, ?, ?], [?, ?, ?, ?, ?], [?, ?, ?, ?, ?]]$

Instance 2

$S = ['many', 'big', 'no', 'affordable', 'one']$

$G = [['many', ?, ?, ?, ?], [?, 'small', ?, ?, ?], [?, ?, 'affordable', ?, ?], [?, ?, ?, ?, 'one']]$

$S = ['many', 'big', 'no', 'affordable', 'one']$

Instance 3

$['many', 'big', 'no', 'affordable', 'one']$

$G = [['many', ?, ?, ?, ?]]$

$S = ['many', ?, 'no', ?, ?]$

Instance 4 'many', 'small', 'no', 'affordable', 'many'

$S = ['many', ?, 'no', ?, ?]$

Instance 5

$G = [['many', ?, ?, ?, ?]]$

$S = ['many', ?, 'no', ?, ?]$

Gradient Descent in Machine Learning

Area (Acre sq)	Price(in millions)
0.5	1.4
2.3	1.9
2.9	3.2

Every machine learning algorithm uses an optimization algorithm to minimize its cost function. Let us take an example where gradient descent is used to optimize a linear regression problem. Consider the data given below that shows dataset of various houses depending upon their area.

Figure 1 shows this data plotted on a graph. To fit the best-fit line, we have to optimize the slope and the intercept of the line. In the explanation given below, we have assumed a value of 0.64 as the slope of the line.

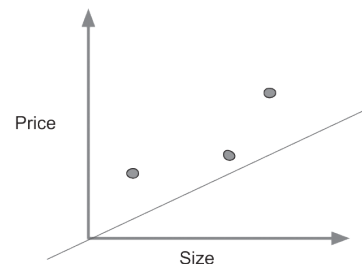


Figure 1 Linear Regression Line for Sample Data

Step 1: Calculate the residual errors for each data point. For doing this, Guess a value for the intercept. Here, we have assumed zero. In linear regression,

$$\text{Predicted value} = \text{intercept} + \text{slope} * x$$

Hence, predicted value = $0 + 0.64 * 0.5 = 0.32$

Squared residual error for each point is then computed as,

$$\text{Squared Residual error} = (\text{actual error} - \text{predicted})^2$$

For example, for the first point, squared residual error = $(1.4 - 0.32)^2 = (1.1)^2$

Find this value for all the points and then calculate the sum of squared error. Plot this point in a graph. Value of the intercept is marked on the x-axis and sum of squared residual error is plotted on the y-axis.

We can plot points for different values of intercept as shown in Figure 2.

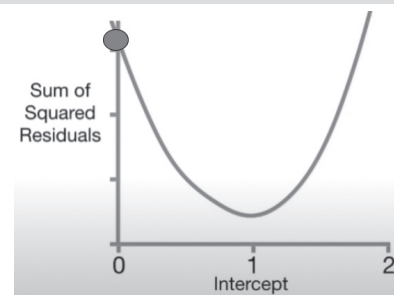


Figure 2 Graph Depicting Sum of Squared Residuals vs Intercepts

A.6 Data Science and Machine Learning Using Python

The main aim of using the Gradient Descent algorithm here is to find the minimum of this cost function. This is done by finding derivatives with respect to the intercept using the equation given below.

$$\begin{aligned}\text{Derivative} &= d/d(\text{intercept}) (1.4 - (\text{intercept} + 0.64 * 0.5))^2 \\ &+ d/d(\text{intercept}) (1.9 - (\text{intercept} + 0.64 * 2.3))^2 \\ &+ d/d(\text{intercept}) (3.2 - (\text{intercept} + 0.64 * 2.9))^2\end{aligned}$$

Like this, derivative of each term is calculated individually and then added. Note that since the slope is taken as a constant value, its derivative is zero.

Set the value of intercept to 0 to find the next value of intercept. Gradient descent subtracts the step size from the current value of intercept to get the new value of intercept. This step size is calculated by multiplying the derivative to the learning rate. Usually, we take the value of the learning rate to be 0.1, 0.01 or 0.001. The value of step should not be too big to skip the minimum point (Figure 3).

Therefore, setting the learning rate to 0.1, step size is equal and is calculated as,

$$\begin{aligned}\text{Step size} &= -0.57 * 0.1 \text{ (where derivative, } d = -0.57\text{)} \\ \text{New intercept} &= \text{old intercept} - \text{step size} \\ &= 0 - (-0.57) = 0.57\end{aligned}$$

Putting new intercept in the derivative function, we get,

$$\begin{aligned}d \text{ sum of squared error} / d(\text{intercept}) &= -2 (1.4 - (0.57 + 0.64 * 0.5)) + \\ &-2 (1.9 - (0.57 + 0.64 * 2.3)) + -2 (3.2 - (0.57 + 0.64 * 2.9)) \\ &= -2.3\end{aligned}$$

Next step size

$$\begin{aligned}\text{Step size} &= -2.3 * 0.1 \\ \text{New intercept} &= \text{old intercept} - \text{step size} \\ &= 0.57 - (-0.23) = 0.8\end{aligned}$$

Repeat the process to find a new value of the intercept. Check out yourself, you will get derivative value = -0.9, step size as -0.9×0.1 and new intercept as 0.89.

From the above calculation, we can conclude that the value of the step is high when the optimal solution is far away and it is less when we approach an optimal solution. This clearly states that gradient descent takes a bigger step when it is away from the solution and takes small steps when nearer to an optimal solution.

If we have more than one parameter, then the process remains the same but the number of derivatives increases. Also, here we have used the sum of squared residuals, but we could have used any other loss function as well such as least squares.

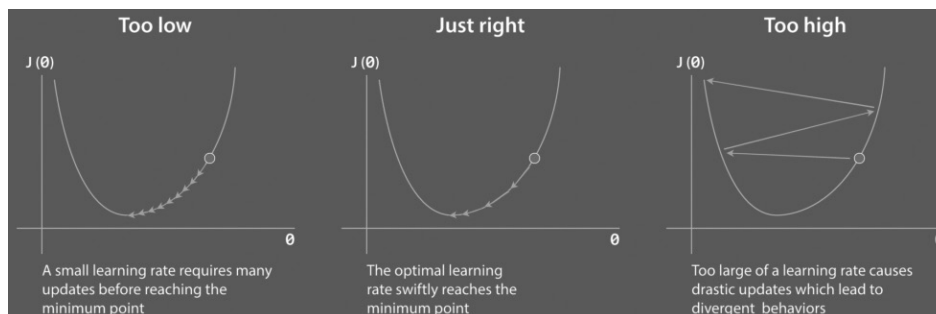
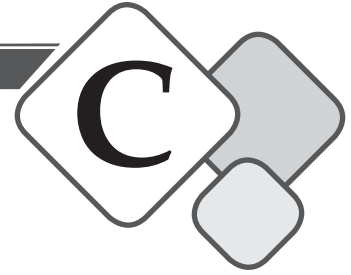


Figure 3 Learning Rate vs Intercept



Markov Decision Process in Reinforcement Learning

In some applications we want the machine learning algorithm to determine whether a new data value belongs to the same distribution as existing values (*inlier*), or to a different one (*outlier*). Usually, outlier detection is used to clean data sets, identify misleading values and take necessary action to deal with them. However, we must consider two important terminology here – outlier detection and novelty detection. Two important distinctions must be made:

Outlier detection aims to identify data points that are far from the others. The algorithm tries to fit regions where training data is most concentrated thereby ignoring the deviant points.

Novelty detection is used to determine whether a new data point is an outlier or not.

Therefore, both outlier detection and novelty detection are used to detect anomalies. While, outlier detection uses unsupervised anomaly detection techniques, novelty detection, on the other hand, uses semi-supervised technique. Outliers are usually located in low-density regions and thus do not form dense clusters. On the contrary, novelties form dense cluster(s) in low-density regions of the training data that are considered as normal in this context.

1. Use the housing.csv data set containing 13 input variables to describe the properties of the house and suburb. Predict the median value of houses in the suburb in thousands of dollars and print the MAE value.
2. Use isolation forest to detect and remove outliers and then again measure the MAE (Mean Absolute Error).
3. Apply the Minimum Covariance Determinant function and see if the MAE gets reduced or not.
4. Also print MAE by using one classification SVM algorithm.

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
from sklearn.ensemble import IsolationForest
from sklearn.covariance import EllipticEnvelope
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
# load the dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
```



```

df = read_csv(url, header=None)
# retrieve the array
data = df.values
# split into input and output elements
X, y = data[:, :-1], data[:, -1]
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=1)
# fit the model
model = LinearRegression()
model.fit(X_train, y_train)
# evaluate the model
yhat = model.predict(X_test)
# evaluate predictions
mae = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % mae)

```

Note that Isolation Forest (or iForest) is a tree-based anomaly detection algorithm that models the normal data in such a way that anomalies that are both few in number and different in the feature space are isolated.

It is usually preferred to detect outliers in high-dimensional datasets. The algorithm ‘isolates’ data points by randomly selecting a feature, and then randomly selecting a split value between the maximum and minimum values of the selected feature. This recursive partitioning is represented using a tree structure.

The number of splits required to isolate a sample is equivalent to the path length from the root node to the leaf node. This path length, when averaged over a forest of such random trees, is used as a measure of normality and validates the decision function. Thus, random partitioning generates shorter paths for anomalies.

To implement Isolation Forest, scikit-learn library has the `IsolationForest` class. It also has a “contamination” argument, which is used to help estimate the number of outliers in the dataset. Its value usually varies from 0.0 to 0.5. By default, it is set to 0.1.

```

# identify outliers in the training dataset
iso = IsolationForest(contamination=0.1)
yhat = iso.fit_predict(X_train)
# select all rows that are not outliers
mask = yhat != -1
X_train, y_train = X_train[mask, :], y_train[mask]
# fit the model
model = LinearRegression()
model.fit(X_train, y_train)
# evaluate the model
yhat = model.predict(X_test)
# evaluate predictions
mae = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % mae)

```


Minimum covariance determinant

If the input variables have a Gaussian distribution, then simple statistical methods can be used to detect outliers. However, an efficient implementation of this technique for multivariate data is known as the Minimum Covariance Determinant (MCD).

The scikit-learn library has the `EllipticEnvelope` class that has a “*contamination*” argument. Value of this argument can be set after a little trial and error.

```
from pandas import read_csv
from sklearn.covariance import EllipticEnvelope
# identify outliers in the training dataset
ee = EllipticEnvelope(contamination=0.01)
yhat = ee.fit_predict(X_train)
# select all rows that are not outliers
mask = yhat != -1
X_train, y_train = X_train[mask, :], y_train[mask]
# fit the model
model = LinearRegression()
model.fit(X_train, y_train)
# evaluate the model
yhat = model.predict(X_test)
# evaluate predictions
mae = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % mae)
```

Local outlier factor

A simple approach to identifying outliers is to locate data points that are far from the other data points. This works well with low dimensionality (few features), but becomes less reliable as the number of features increases.

The local outlier factor (LOF) is a technique that uses nearest neighbors for outlier detection. Each data point is given a score based on how isolated or how likely it is to be an outlier with respect to its local neighborhood. Data points with the largest score are more likely to be outliers.

The strength of LOF algorithm is that it considers local as well as global properties of datasets to perform well even on those datasets that have abnormalities present in different densities. For this, besides looking into how isolated a data point is, how isolated it is with respect to its surrounding neighborhood is also given due consideration.

The `LocalOutlierFactor` class in the scikit-learn library provides an implementation of this outlier detection technique. The function accepts the “*contamination*” argument that indicates the expected percentage of outliers in the dataset. Its default value is 0.1.

```
# identify outliers in the training dataset
lof = LocalOutlierFactor()
yhat = lof.fit_predict(X_train)
# select all rows that are not outliers
mask = yhat != -1
X_train, y_train = X_train[mask, :], y_train[mask]
# fit the model
```


A.10 Data Science and Machine Learning Using Python

```
model = LinearRegression()
model.fit(X_train, y_train)
# evaluate the model
yhat = model.predict(X_test)
# evaluate predictions
mae = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % mae)
```

One-class SVM

The support vector machine (SVM) algorithm for binary classification when used to capture density of the majority class and classify new data points based on the extremes of the density function as outliers, is referred to as One-Class SVM. The one-class SVM discovers outliers in input data for both regression and classification datasets.

The `OneClassSVM` class in the scikit-learn library provides an implementation of the one-class SVM algorithm. This class provides the “*nu*” argument that specifies the approximate ratio of outliers in the dataset. By default, its value is 0.1. However, we can set it to any other value also by a little trial and error.

```
# identify outliers in the training dataset
ee = OneClassSVM(nu=0.01)
yhat = ee.fit_predict(X_train)
# select all rows that are not outliers
mask = yhat != -1
X_train, y_train = X_train[mask, :], y_train[mask]
# fit the model
model = LinearRegression()
model.fit(X_train, y_train)
# evaluate the model
yhat = model.predict(X_test)
# evaluate predictions
mae = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % mae)
```

OUTPUT

```
Without Outlier Detection, MAE: 3.417
IsolationForest MAE: 3.211
LocalOutlierFactor MAE: 3.229
EllipticEnvelope MAE: 3.232
OneClassSVM MAE: 3.211
```


Outlier Detection

Principal component analysis (PCA)

Machine Learning algorithms give their best performance when the training dataset is large and concise. Though huge amount of data is the backbone of any predictive model, using such a large data set has its own pitfalls – the biggest one being the curse of dimensionality.

In a large dimensional dataset, there may exist many redundant features or multiple inconsistencies in the features. This not only increases the computation time but also makes data processing and exploratory analysis more convoluted. To overcome these issues, we perform dimensionality reduction techniques to filter only a limited set of significant features needed for training the model.

Principal components analysis is a dimensionality reduction technique that identifies correlations and patterns in a data set to transform it to a significantly lower dimension data set without losing any important information. It is usually performed to solve complex data-driven problems having very high number of dimensions. Follow the steps given below to perform dimensionality reduction using PCA:

Step 1: Standardize the data.

Step 2: Calculate the covariance matrix.

Step 3: Compute the eigenvectors and eigenvalues.

Step 4: Identify the Principal Components.

Step 5: Reduce the dimensions of the dataset.

Step I: To standardize data, we need to scale the data in such a way that values of all the variables lie within a similar range. For example, if there are two variables – one having values ranging between 100 and 1000, and the other having values from just 10 to 100. In such a scenario, the output produced using these predictor variables will be highly biased. Variable with a larger range will impact the outcome in a bigger way. Therefore, data must be standardized into a comparable range. This is done by subtracting each value in the data from the mean and dividing it by the overall deviation in the dataset as given below.

$$Z = \frac{\text{Variable value} - \text{mean}}{\text{Standard deviation}}$$

Step 2: To identify the correlation and dependencies among the features in the data set. This step is essential because strongly dependent variables contain biased and redundant information thereby affecting the overall performance of the model.

Mathematically, a covariance matrix is a $p \times p$ matrix, where p is the number of dimensions in the data set. Each entry in the matrix represents the covariance of the corresponding variables. We can visualize the covariance matrix for variables a and b as,

$$\begin{bmatrix} \text{Cov}(a, a) & \text{Cov}(a, b) \\ \text{Cov}(b, a) & \text{Cov}(b, b) \end{bmatrix}$$

Note that,

- $\text{Cov}(a, a)$ is the covariance of a variable with itself.
- $\text{Cov}(a, b)$ is the covariance of the variable ' a ' with respect to the variable ' b '.
- $\text{Cov}(a, b) = \text{Cov}(b, a)$
- If $\text{Cov}(a, b) < 0$, the respective variables are indirectly proportional to each other.
- If $\text{Cov}(a, b) > 0$, the respective variables are directly proportional to each other.

Step 3: Find the covariance matrix to determine the principal components of the dataset.

Remember that, Principal Components are the new set of variables that are obtained from the initial set of variables. These variables are highly significant and independent of each other. Principal components compress and possess most of the useful information that was scattered among the initial variables. For example, if the data set has 5 dimensions, then 5 principal components are computed in such a way that the first principal component stores the maximum possible information, the second stores the remaining maximum information, so on and so forth.

Eigenvectors and eigenvalues are always computed as a pair. This means that for every eigenvector there is an eigenvalue. Number of eigenvectors depends on the number of dimensions in the data. For example, in a 2-dimensional dataset, two eigenvectors are computed.

By looking at the Covariance matrix, we need to understand where in the data there is maximum variance. More variance gives more information about the data. Thus, *eigenvectors and eigenvalues will then be used to compute the Principal Components of the dataset.*

Step 4: After computing the Eigenvectors and eigenvalues, we need to arrange these in the descending order because the eigenvector with the highest eigenvalue is the most significant and thus forms the first principal component. The principal components of lesser significances can be removed in order to reduce the dimensions of the data. Finally, a feature matrix containing all the significant data variables that possess maximum information about the data is created.

Step 5: The last step in the PCA technique is to reduce the dimensions of the data set. This is done by re-arranging the original data with the final principal components having maximum and the most significant information of the data set.

To replace the original data axis with the newly formed Principal Components, we need to multiply the transpose of the original data set by the transpose of the obtained feature vector.

Technically, PCA can be explained as given below.

- PCA reduces dimensions of data by projecting it onto lines drawn through data. Line that goes through the data in the direction of the greatest variance is drawn first. This is calculated by computing the eigenvectors of the covariance matrix.
- To represent linear transformation through a matrix, we multiply some data points by the matrix to move around the graph.

- The eigenvectors of a linear transformation are the lines where if a data point started on that line, they end on that line, and the eigenvalue says how far they have moved. Thus, eigenvectors talk about the direction of a linear transformation.
- A covariance matrix can be used as a linear transformation, and the square root of the covariance matrix of some data will perform a linear transformation that moves Gaussian data points into the shape of our data having the same standard deviations and correlations as the data.
- So, the eigenvectors of the square root of the covariance matrix tells us in what direction data has been smeared away from perfectly normally distributed and the eigenvalues tell us how far. The direction of the smearing is the same as the principal components, and how far our data was smeared tells us which direction has the greatest variance.

Example: Implementing PCA on height_weight dataset

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.decomposition import PCA
from scipy.linalg import sqrtm

#Load the data

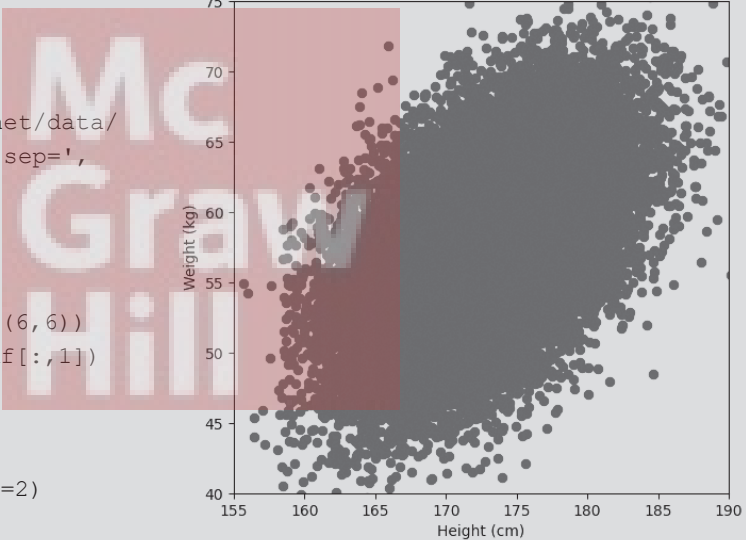
df=pd.read_csv('http://www.billconnelly.net/data/height_weight.csv', sep=',', header=None)
df = np.array(df)

#Plot the data
fig = plt.figure(figsize=(6,6))
plt.scatter(df[:,0], df[:,1])
ax = plt.gca()

#Perform the PCA
pca = PCA(n_components=2)
pcafit = pca.fit(df)

'''#Display the components
for comp, ex in zip(pcafit.components_, pcafit.explained_variance_):
    rotation_matrix = np.array([[ -1, 0], [0, -1]])
    ax.annotate(' ', pcafit.mean_ + np.sqrt(ex)*comp, pcafit.mean_,
        arrowprops={'arrowstyle':'->', 'linewidth':2})
    ax.annotate(' ', pcafit.mean_ + rotation_matrix@(np.sqrt(ex)*comp), pcafit.mean_,
        arrowprops={'arrowstyle':'->', 'linewidth':2})
'''

plt.xlabel("Height (cm)")
plt.ylabel("Weight (kg)")
plt.xlim((155,190))
plt.ylim((40,75))
plt.show()
```

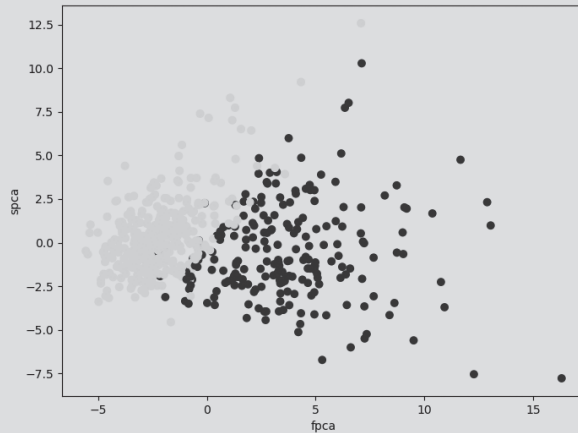


Example: Implementing PCA on breast cancer data set

```

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.datasets
    import load_breast_cancer
cancer = load_breast_cancer()
cancer.keys()
df = pd.DataFrame(cancer['data'],
columns=cancer['feature_names'])
print(df.head())
#PCA Visualization
from sklearn.preprocessing
    import StandardScaler
scaler = StandardScaler()
scaler.fit(df)
scaled_data = scaler.transform(df)
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(scaled_data)
x_pca = pca.transform(scaled_data)
print("SHAPE OF PCA DATA = ", x_pca.shape)
plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0],x_pca[:,1],c=cancer['target'],cmap='plasma')
plt.xlabel('fpca')
plt.ylabel('spca')
plt.show()

```



	mean radius	mean texture	...	worst symmetry	worst fractal dimension
0	17.99	10.38	...	0.4601	0.11890
1	20.57	17.77	...	0.2750	0.08902
2	19.69	21.25	...	0.3613	0.08758
3	11.42	20.38	...	0.6638	0.17300
4	20.29	14.34	...	0.2364	0.07678

[5 rows x 30 columns]

SHAPE OF PCA DATA = (569, 2)

Principal Component Analysis

Word Cloud is a data visualization technique that is used for representing textual data in such a way that size of each word indicates its frequency or importance. Significant textual data are highlighted in a word cloud. Word clouds are widely used for analyzing data from social network websites.

Forming a word cloud in Python

To generate a word cloud in Python, modules like matplotlib, pandas and wordcloud should be installed. To install these packages, just type the following commands on the Windows Command Prompt.

```
pip install matplotlib
pip install pandas
pip install wordcloud
```

Let us form a word cloud comprising YouTube comments on videos of popular artists. The data-set is available on <https://archive.ics.uci.edu/ml/machine-learning-databases/00380/>

```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd
df = pd.read_csv(r"Youtube04-Eminem.csv", encoding="latin-1")
comment_words = ''
stopwords = set(STOPWORDS)
# iterate through the csv file
for word in df.CONTENT:
    # typecaste each val to string
    word = str(word)
    # split the value
    tokens = word.split()
    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()
    comment_words += " ".join(tokens)+" "
```



```
wordcloud = WordCloud(width = 800, height = 800,
                      background_color = 'white',
                      stopwords = stopwords,
                      min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



Advantages of word clouds

1. Used to analyze feedback.
2. Identify Search Engine Optimization (SEO) keywords for digital marketing.

Limitations of word clouds

1. Word Clouds are not equally useful in every situation.
2. Data should be optimized for context.

Web Scraping

Web scraping is an automated technique used to pull a large amount of data from websites. Data on the websites is usually unstructured. So, web scraping is used to extract this unstructured data so that it can be stored in a structured form as shown in Figure 1.

Programming Tip:

Some websites do not allow web scraping



Figure 1 Generating Structured Data From Web Pages Using Web Scraping

Structured data collected from websites is very useful for the following purpose.

Services like ParseHub use web scraping technique to extract large amount of data from online shopping websites to **compare the prices of products**.

Companies that make extensive use of email marketing, apply web scraping to collect email ID to **send bulk emails** for promoting their products.

Social Media Scraping uses web scraping to collect data from Social Media websites like Twitter to understand latest business trends.

Research and Development team uses web scraping to collect a large set of data including important statistics, general information, temperature, etc. from websites for analysis to interpret useful results.

Job-related websites use web scraping to collect details regarding job openings, interviews from different websites to list them on a single platform to make it easily accessible to the user.

Implementing web scraping through Python

When web scraping code is executed, a request is sent to the specified URL. As a response to the request, the server sends the data and gives the permission to read the HTML or XML page. The code then parses the web page to locate and extract the required data. Steps involved in scraping the web through Python are given below.

Step 1: Locate the URL that is to be scraped.

Step 2: Parse the web page.

Step 3: Find the required data.

Step 4: Extract the data.

Step 5: Store the data in the required format.

Let us scrap the Wikipedia website. But before that install the *urllib* and *BeautifulSoup* modules.

Note before printing the text, we will clean it by removing the headers surrounded by '==' and replace '\n' with '' (an empty string) by writing `text.replace('\n', '')`.

```
# Import packages
import wikipedia
# Specify the title of the Wikipedia page
wiki = wikipedia.page('Natural_language_processing')
# Extract the plain text content of the page, excluding images, tables, and
other data.
text = wiki.content
# Replace '==' with '' (an empty string)
text = text.replace('==', '')
# Replace '\n' (a new line) with '' & end the string at $1000.
text = text.replace('\n', '')[:12]
print(text)
```

SAMPLE OUTPUT

Natural language processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data. The result is a computer capable of "understanding" the contents of documents, including the contextual nuances of the language within them. The technology can then accurately extract information and insights contained in the documents as well as categorize and organize the documents themselves. Challenges in natural language processing frequently involve speech recognition, natural language

Remember

To know whether a website allows web scraping or not, just look at the website's "robots.txt" file. You can find this file by appending "/robots.txt" to the URL that you want to scrape

understanding, and natural-language generation. History Natural language processing has its roots in the 1950s. Already in 1950, Alan Turing published an article titled "Computing Machinery and Intelligence" which proposed what is now called the Turing test as a criterion of intelligence, a task that involves the automated interpretation and generation of natural language, but at the time not articulated as a problem separate from artificial intelligence. = Symbolic NLP (1950s - early 1990s) =The premise of symbolic NLP is well-summarized by John Searle's Chinese room experiment: Given a collection of rules (e.g., a Chinese phrasebook, with questions and matching answers), the computer emulates natural language understanding (or other NLP tasks) by applying those rules to the data it is confronted with. 1950s: The Georgetown experiment in 1954 involved fully automatic translation of more than sixty Russian sentences into English.

Plotting the word cloud

Let us quickly look into the details of the `Wordcloud()` function before using it to plot most frequent words in our sample text. Some important arguments of this function are:

width/height: Specify the width and height of the word cloud dimension.

random_state: The value of this parameter may alter the word cloud formed. By setting its value to 1, you ensure that every time you run the same script on the same input data, the same word cloud is formed. Correspondingly, experiment with this value until you find the one that results in the word cloud that best suits your needs.

background_colour specifies the background color of the word cloud formed. Usually, it is set to 'white' and 'black'.

colormap is used to set up the color theme that the words are displayed in. Some commonly used colormaps are 'rainbow', 'seismic', 'Pastel1' and 'Pastel2'.

collocations is set to *False* to ensure that the word cloud does not appear to contain duplicate words. For example, if *collocations* is *True* then 'web', 'scraping' and 'web scraping' appear as a collocation in the word cloud, giving an impression that words have been duplicated.

stopwords are common words which are irrelevant to the meaning of the text. For example, 'we', 'are' and 'the' are some stopwords.

Note that you can see the list of stopwords, by writing `print(STOPWORDS)`. To replace a stopword with another one, you can write, `STOPWORDS.update(['word1', 'word2'])`.

Displaying the Word Cloud from text obtained by scraping Wikipedia

```
# Import packages
import wikipedia
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS

def plot_cloud(wordcloud):
    # Set figure size
    plt.figure(figsize=(40, 30))
    # Display image
```


cloud as a .png file, we can write
cloud, we can use an image as a ma
n.
that takes text from a txt file. Our
ot as plt
WordCloud, STOPWORDS

Copyright © 2023 by McGraw Hill Education (India) Private Limited



HTML and XML documents by crawling and analysis.

Part Website to extract the Price, Name of the desired webpage. Then, inspect the page is inspected to see the content, this, select the particular information. "Browser Inspector Box" will be opened to extract data represented using HTML.

[rooms.github.io/webscraper-py](https://github.com/rooms.github.io/webscraper-py)

Pandas for data manipulation and analysis.

For example, to scrap Flipkart Website to extract the Price, Name, and Rating of Laptops, we will first use its URL to open the desired webpage. Then, inspect the webpage. Data is usually presented using nested tags. So, the page is inspected to see the particular tag under which the data to be scraped is nested. For this, select the particular information on the web page, right click on it and select Inspect. The “Browser Inspector Box” will be opened and name of the tag will be displayed at the top.

Copyright © 2023 by McGraw Hill Education (India) Private Limited

OUTPUT

```
[ 'Test Sites', 'E-commerce training site'] 7 reviews
From the URL https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/, extract and store top_items in a list.
import requests
from bs4 import BeautifulSoup
# Make a request
page = requests.get(
    "https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/")
soup = BeautifulSoup(page.content, 'html.parser')
# Create top_items as empty list
top_items = []
# Extract and store in top_items according to instructions on the left
products = soup.select('div.thumbnail')
for elem in products:
    title = elem.select('h4 > a.title')[0].text
    review_label = elem.select('div.ratings')[0].text
    info = {
        "title": title.strip(),
        "review": review_label.strip()
    }
    top_items.append(info)
print(top_items)
```

OUTPUT

```
[{'title': 'Asus AsusPro Adv...', 'review': '7 reviews'}, {'title': 'Asus ROG Strix G...', 'review': '4 reviews'}, {'title': 'Acer Aspire 3 A3...', 'review': '2 reviews'}]
```

In the above code, text presented using the `div.thumbnail` element, gives a list of individual products. We can iterate over them and use `select` to get the title and other details. When we write, `h4 > a.title`, selector would return a list. The list's 0th element is used to extract the text. The text is stripped to remove any extra whitespace before appending it to the list.

From the URL <https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/>, **extract and store top_items in a list using the img tag.**

```
import requests
from bs4 import BeautifulSoup
# Make a request
page = requests.get(
    "https://codedamn-classrooms.github.io/webscraper-python-codedamn-classroom-website/")
soup = BeautifulSoup(page.content, 'html.parser')
# Create top_items as empty list
image_data = []
```



```
# Extract and store in top_items
images = soup.select('img')
for image in images:
    src = image.get('src')
    alt = image.get('alt')
    image_data.append({"src": src, "alt": alt})
print(image_data)
```

OUTPUT

```
[{'src': '/webscraper-python-codedamn-classroom-website/logo_white.svg', 'alt': 'Web Scraper'}, {'src': '/webscraper-python-codedamn-classroom-website/cart2.png', 'alt': 'item'}, {'src': '/webscraper-python-codedamn-classroom-website/cart2.png', 'alt': 'item'}, {'src': '/webscraper-python-codedamn-classroom-website/cart2.png', 'alt': 'item'}, {'src': '/webscraper-python-codedamn-classroom-website/fbicon.png', 'alt': 'Web Scraper on Facebook'}, {'src': '/webscraper-python-codedamn-classroom-website/twicon.png', 'alt': 'Web Scraper on Twitter'}]
```

Write a code to extract the href attribute of links with their text from the URL <http://forecast.weather.gov/MapClick.php?lat=37.7772&lon=-122.4168>.

```
import requests
from bs4 import BeautifulSoup
# Make a request
page = requests.get("http://forecast.weather.gov/MapClick.php?lat=37.7772&lon=-122.4168")
soup = BeautifulSoup(page.content, 'html.parser')
# Create top_items as empty list
all_links = []

# Extract and store in top_items according to instructions on the left
links = soup.select('a')
for ahref in links:
    text = ahref.text
    text = text.strip() if text is not None else ''

    href = ahref.get('href')
    href = href.strip() if href is not None else ''
    all_links.append({"href": href, "text": text})

print(all_links)
```

OUTPUT

```
[{'href': 'http://www.noaa.gov', 'text': ''}, {'href': 'http://www.weather.gov', 'text': ''}, {'href': 'http://www.commerce.gov', 'text': ''}, {'href': 'http://www.weather.gov', 'text': 'HOME'}]
```

Note that we can also form a dictionary using or more list elements to get complete details (name, model, price, year of manufacturing, etc.) of all the products. For example, by writing,


```
# Create top_items as empty list
all_products = []
# Extract and store in top_items
products = soup.select('div.thumbnail')
for product in products:
    name = product.select('h4 > a')[0].text.strip()
    description = product.select('p.description')[0].text.strip()
    price = product.select('h4.price')[0].text.strip()
    reviews = product.select('div.ratings')[0].text.strip()
    image = product.select('img')[0].get('src')
    all_products.append({
        "name": name,
        "description": description,
        "price": price,
        "reviews": reviews,
        "image": image
    })
```

We can extract data from all the elements and attributes. The code given below takes another example; it extracts data from <http://forecast.weather.gov/MapClick.php?lat=37.7772&lon=-122.4168> to give the weather information of a particular location. The collected data is then stored in a data frame for efficient analysis.

```
import requests
import pandas as pd
from bs4 import BeautifulSoup
# Make a request
page=requests.get("http://forecast.weather.gov/MapClick.php?lat=37.7772&lon=-122.4168")
soup = BeautifulSoup(page.content, 'html.parser')
seven_day = soup.find(id="seven-day-forecast")
forecast_items = seven_day.find_all(class_="tombstone-container")
tonight = forecast_items[0]
print(tonight.prettify())
period = tonight.find(class_="period-name").get_text()
short_desc = tonight.find(class_="short-desc").get_text()
temp = tonight.find(class_="temp").get_text()
print(period)
print(short_desc)
print(temp)
period_tags = seven_day.select(".tombstone-container .period-name")
periods = [pt.get_text() for pt in period_tags]
short_descs = [sd.get_text() for sd in seven_day.select(".tombstone-container .short-desc")]
temps = [t.get_text() for t in seven_day.select(".tombstone-container .temp")]
descs = [d["title"] for d in seven_day.select(".tombstone-container img")]
weather = pd.DataFrame({
```


A.24 Data Science and Machine Learning Using Python

```
"period": periods,
"short_desc": short_descs,
"temp": temps,
"desc":descs
})
print(weather)
```

OUTPUT

	period	...	desc
0	Overnight	...	Overnight: Mostly clear, with a low around 47....
1	Saturday	...	Saturday: Sunny, with a high near 71. Calm win...
2	SaturdayNight	...	Saturday Night: Mostly clear, with a low aroun...
3	Sunday	...	Sunday: Sunny, with a high near 67. Light west...
4	SundayNight	...	Sunday Night: Mostly clear, with a low around ...
5	Monday	...	Monday: Sunny, with a high near 65.
6	MondayNight	...	Monday Night: Clear, with a low around 49.
7	Tuesday	...	Tuesday: Sunny, with a high near 72.
8	TuesdayNight	...	Tuesday Night: Clear, with a low around 51.

[9 rows x 4 columns]

Mc
Graw
Hill

Web Scrapping and Making Word Cloud

Often while analyzing data, we need to handle temporal values. Python deals with different types of date and time formats gracefully. For this, the **datetime** library has all necessary methods and functions that can be used for representing date and time values, performing arithmetic operations on them and comparing two date time values.

Date time representation

A date and its parts are represented using different datetime functions. Format specifiers are often used to display different information like name of the month, date or week day. The code given below displays today's date and its components.

```
import datetime
print('Current Date and Time Today is :', datetime.datetime.today())
date_today = datetime.date.today()
print('Today\'s Date : ', date_today)
print('Current Year :', date_today.year)
print('Current Month :', date_today.month)
print('Current Month Name:', date_today.strftime('%B'))
print('Week Day :', date_today.day)
print('Week Day Name:', date_today.strftime('%A'))
```

OUTPUT

```
Current Date and Time Today is : 2021-03-16 20:59:06.752614
Today's Date : 2021-03-16
Current Year : 2021
Current Month : 3
Current Month Name: March
Week Day : 16
Week Day Name: Tuesday
```

Date time arithmetic

To perform calculations on date and time values, we can store these values using variables and then apply the required mathematical operator on these variables. Code given below adds or subtracts two values from date objects.

A.26 Data Science and Machine Learning Using Python

```
import datetime
day1 = datetime.date(2020, 8, 26)
print('Day1 : ', day1.ctime())

day2 = datetime.date(2021, 3, 16)
print('Day2 : ', day2.ctime())

# Find the difference between the dates
print('Number of Days between two dates : ', day1-day2)

date_today = datetime.date.today()

# Create a delta of Ten Days
no_of_days = datetime.timedelta(days = 10)

# Use Delta for Past Date

before_ten_days = date_today - no_of_days
print('Date Before Ten Days : ', before_ten_days)

# Use Delta for future Date

after_ten_days = date_today + no_of_days
print('Date After Ten Days : ', after_ten_days)
```

OUTPUT

```
Day1 :  Wed Aug 26 00:00:00 2020
Day2 :  Tue Mar 16 00:00:00 2021
Number of Days between two dates :  -202 days, 0:00:00
Date Before Ten Days :  2021-03-06
Date After Ten Days :  2021-03-26
```

Date time comparison

Two date and time values can be easily compared using logical operators. In the code given below, two dates are compared using logical operators.

```
import datetime

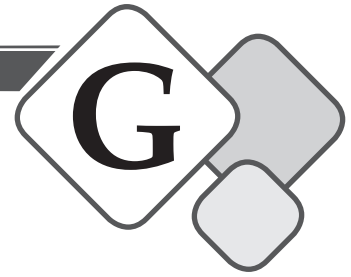
day1 = datetime.date(2021, 8, 26)
print('Day1 : ', day1.ctime())

day2 = datetime.date(1954, 4, 26)
print('Day2 : ', day2.ctime())

if day1 > day2:
    print("Day 2 occurred before Day 1")
else:
    print("Day 1 occurred before Day 2")
```

OUTPUT

```
Day1 :  Thu Aug 26 00:00:00 2021
Day2 :  Mon Apr 26 00:00:00 1954
Day 2 occurred before Day 1
```

Working with Date and Time

Association analysis is an excellent data analysis tool that is used for Market Basket Analysis. The term Market Basket Analysis is often used to find common patterns of items in large datasets.

For example, the information that people who buy bread and butter also buy eggs in the same transaction, helps the super market to offer attractive discounts on the purchase of eggs. Such an offer would increase the likelihood of a customer buying all the three items in a single visit to the super market.

Association rules

Association rules allow users to see the similarities between items. When association rule mining is applied on a dataset, we get association rules stating “if this, then that”. These rules can then be used for:

- Recommending products. For example, Amazon uses association rules to learn that customers who purchased product A also bought product B.
- Recommend music or recommend songs by a particular artist.
- To diagnose a patient to learn whether he has a particular disease or not.
- Content optimization to provide relevant content in a magazine, website, or blog.

Understanding association rule mining

Just imagine, can anyone analyze thousands of receipts that represent transactions to learn about the items that were purchased together? No, not at all. But this can be done very easily in Python. We can take a retail dataset that stores receipts (or transactions) as rows and the items purchased as columns to understand how association rule mining is done.

In association rule mining, items that are purchased together forms the item set I. Here, $I = \{i_1, i_2, \dots, i_n\}$

A transaction is represented as $t_n = \{i_1, i_k, \dots, i_n\}$

A rule that has LHS and an RHS represents which item(s) are frequently bought and with which other items. For example, if we say, itemset A = itemset B, then items in itemset B are frequently purchased along with items in the itemset A.

The notation, $\{i_1, i_2\} \Rightarrow \{i_k\}$ signifies that if a customer buys an item in the item set on the LHS, then he/she is likely to buy the item given on the RHS. Therefore, if we write, $\{\text{bread, butter}\} \Rightarrow \{\text{milk}\}$, then it means that a customer buying bread and butter is likely to buy milk in the same transaction.

Strength of an association rule can be measured using three parameters that are given below.

Support: It is calculated as the fraction of which an item occurs in dataset. This means that, support of an item points to the popularity of the item in the transaction set. Higher the support, more popular is the item. Mathematically, support can be expressed as,

$$\text{Support} = \text{Number of transactions with both A and B} = P(A \cup B)$$

Confidence: Confidence gives the conditional probability that a rule is correct for a new transaction with items on the LHS. This means that confidence specifies that a customer who buys item A will also buy item B. Therefore, confidence measures how often item B is purchased when an item A is bought. Mathematically, confidence is written as,

$$\text{Confidence}(A \Rightarrow B) = \text{Support}(A \text{ and } B) / \text{Support}(A)$$

Association rules with higher confidence indicate that the probability of an item appearing on the RHS is high when the transaction contains items on the LHS.

Lift: Lift is the ratio by which the confidence of a rule exceeds the expected confidence. $\text{lift} = 1$, means that the items on the LHS and RHS are independent of each other. For example, if a customer buys item A, then lift specifies by what percent the likelihood of buying B would increase.

$\text{lift} > 1$, means that the presence of A has increased the probability that item B will also be bought in the same transaction.

$\text{lift} < 1$ indicates that when item A is purchased, the probability that item B will also be purchased in the same transaction decreases.

Mathematically, lift is calculated as,

$$\text{Lift}(A \Rightarrow B) = \text{Confidence}(A \Rightarrow B) / \text{Support}(B)$$

To summarize, $\text{lift} = 1.25$ means that chances of buying item B has increased by 25% in the presence of item A in the same transaction. Therefore, lift indicates the strength of an association rule in the dataset where items A and B occur randomly. In such datasets, lift indicates change in probability of item A in presence of item B. Association rules with $\text{lift} > 1$ are chosen.

For example, if an association rule, $\text{Bread} \Rightarrow \text{Butter}$, where,

$$\text{Prob}(\text{Butter}) = 0.9, \text{Confidence}(\text{Bread} \Rightarrow \text{Butter}) = \text{Prob}(\text{Butter} | \text{Bread}) = 0.75$$

$$\text{Lift}(\text{Bread} \Rightarrow \text{Butter}) = 0.8333.$$

Then, it indicates a negative association between the two items as $\text{lift} < 1$.

Example: 400 customers visited a retail shop. Of these customers, 200 bought Product A, 160 purchased Product B and 100 bought both.

Here, $\text{Support}(\text{Product A}) = 50\%$ as 200 customers bought Product A.

$\text{Support}(\text{Product B}) = 40\%$ as 160 customers bought Product B.

$\text{Support}(\text{Product A and B}) = 25\%$ as 100 customers bought both.

$$\text{Confidence}(\text{Product A, Product B}) = 100/200 = 50\%.$$

This means that a customer who buys A is 50% likely to buy B.

Example: If out of 100 customers, 10 bought milk, 8 bought butter and 4 bought both of them, then calculate support, confidence and limit.

$$\text{support} = P(\text{milk and butter}) = 4/100 = 0.04$$

$$\text{confidence} = \text{support}/P(\text{Butter}) = 0.04/0.08 = 0.50$$

$$\text{lift} = \text{confidence}/P(\text{Milk}) = 0.50/0.10 = 5.0$$

Implementing market basket analysis in Python

For market basket analysis, we will use the Apriori algorithm which is the most popularly used algorithm for extracting frequent item sets. The algorithm uses a bottom-up approach where frequent items are extended one item at a time by testing groups of candidate item sets against the available dataset. This process continues until no further extensions are found. Steps involved in Apriori algorithm include:

Step 1: Specify minimum support and confidence for the association rule.

Step 2: Incorporate all subsets in the transactions that have a support higher than minimum support.

Step 3: Extract rules from these subsets that have confidence higher than minimum confidence.

Step 4: Sort the association rules in the decreasing order of lift.

Step 5: Visualize rules along with confidence and support.

Large retailers like Amazon, Flipkart, etc. use market basket analysis to analyze customer buying habits by finding associations between the different items purchased by customers. Discovery of such associations helps retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers. These strategies include:

- Changing store layout
- Designing catalogue
- Cross marketing on online stores
- Highlighting trending items purchased by customers
- Customized emails to customers who bought specific products offering them discounts on other products that they are likely to buy

Apart from retailers, Market Basket Analysis is also used in the following areas.

- Manufacturing industry uses it for predictive analysis of equipment failure.
- Pharmaceutical and Bioinformatics uses Market Basket Analysis to find out relationships among diagnosis and medicine prescribed to different patient groups.
- Financial and Banking institutions analyze credit card usage data to detect fraudulent cases.
- Market Basket Analysis is also used for analyzing customer behavior by associating purchases with demographic and socio-economic data.

Before implementing association analysis in Python, we must remember that Association rules do not extract preference of an individual. Rather, they find relationships between sets of elements of every distinct transaction. There is a difference between association and recommendation, which can be understood as,

“Frequently Bought Together” Association

“Customers who bought this item also bought” Recommendation

Implementing Apriori algorithm on an online retail dataset that can be downloaded from <http://archive.ics.uci.edu/ml/machine-learning-databases/00352/>

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```


A.30 Data Science and Machine Learning Using Python

```
from apyori import apriori
# Data Preprocessing
dataset = pd.read_csv('Market_Basket_Optimisation.csv', header = None)
print(dataset)

#Getting the list of transactions from the dataset
transactions = []
for i in range(0, 7501):
    transactions.append([str(dataset.values[i,j]) for j in range(0, 20)])

# Training Apriori algorithm on the dataset
rule_list = apriori(transactions, min_support = 0.003, min_confidence = 0.4,
min_lift = 5, min_length = 2)

# Visualizing the list of rules
results = list(rule_list)
for i in results:
    print('\n')
    print(i)
```

OUTPUT

	0	1	...	18	19
0	shrimp	almonds	...	spinach	olive oil
1	burgers	meatballs	...	NaN	NaN
2	chutney	NaN	...	NaN	NaN
3	turkey	avocado	...	NaN	NaN
4	mineral water	milk	...	NaN	NaN
...
7496	butter	light mayo	...	NaN	NaN
7497	burgers	frozen vegetables	...	NaN	NaN
7498	chicken	NaN	...	NaN	NaN
7499	escalope	green tea	...	NaN	NaN
7500	eggs	frozen smoothie	...	NaN	NaN

```
[7501 rows x 20 columns]
RelationRecord(items=frozenset({'whole wheat pasta', 'olive oil',
'mineral water'}), support=0.0038661511798426876, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'whole wheat pasta', 'mineral
water'}), items_add=frozenset({'olive oil'}), confidence=0.402777777777778,
lift=6.115862573099416)])
```

```
RelationRecord(items=frozenset({'whole wheat pasta', 'nan', 'olive oil',
'mineral water'}), support=0.0038661511798426876, ordered_statistics=[Orde
redStatistic(items_base=frozenset({'whole wheat pasta', 'mineral water'}),
items_add=frozenset({'nan', 'olive oil'}), confidence=0.402777777777778,
lift=6.128267973856209), OrderedStatistic(items_base=frozenset({'whole
wheat pasta', 'nan', 'mineral water'}), items_add=frozenset({'olive oil'}),
confidence=0.402777777777778, lift=6.115862573099416)])
```

Copyright © 2023 by McGraw Hill Education (India) Private Limited

Market Basket Analysis

While visualizing data, we often need to look into the geographical component including distribution of a particular phenomenon in a country, region, continent, or any other geo-location of interest. Python provides full support to its developers to generate maps having an extra layer of data representation and visualization. In this section, we have used folium and considered a scenario that displays store locations and net revenue in (\$) of a popular clothing franchise in India. For this task, the data is provided as a sample.CSV file.

Before running the code, make sure that you install the package by writing `pip install folium pandas`.

```
import folium
import pandas as pd
franchises = pd.read_csv('sample.csv')
#view the dataset
print(franchises.head())
center = [27.23559, 75.9061928]
map_india = folium.Map(location=center, zoom_start=8)
for index, franchise in franchises.iterrows():
    location = [franchise['latitude'], franchise['longitude']]
    folium.Marker(location, popup = f'Name:{franchise["store"]}\n Revenue($
):{franchise["revenue"]}') .add_to(map_india)

# save map to html file
map_india.save('index.html')
```

OUTPUT

```
store,latitude,longitude,revenue
Mumbai,19.076090, 72.877426,600000
Lucknow,26.850000,80.949997,120000
Barjora,23.427221,87.287018,30000
Palwal,28.148735,77.332024,40000
```


Run the above code and view the index.html file in the browser.



Maps drawn using folium are interactive. We can zoom in and out by clicking the positive and negative buttons in the top-left corner of the map. We can even drag the map to see different regions. These maps can be customized to reduce the height and width of the map. For this, we can use the branca library that has **Figure** class for resizing. Figure accepts the desired width and height in pixels as its parameters.

```
import folium
import pandas as pd
from branca.element import Figure
fig=Figure(width=1100,height=1000)
m1=folium.Map(width=1100,height=1000,location=[19.4540,
68.345],zoom_start=5,min_zoom=2,max_zoom=7)
fig.add_child(m1)
# save map to html file
m1.save('index1.html')
```



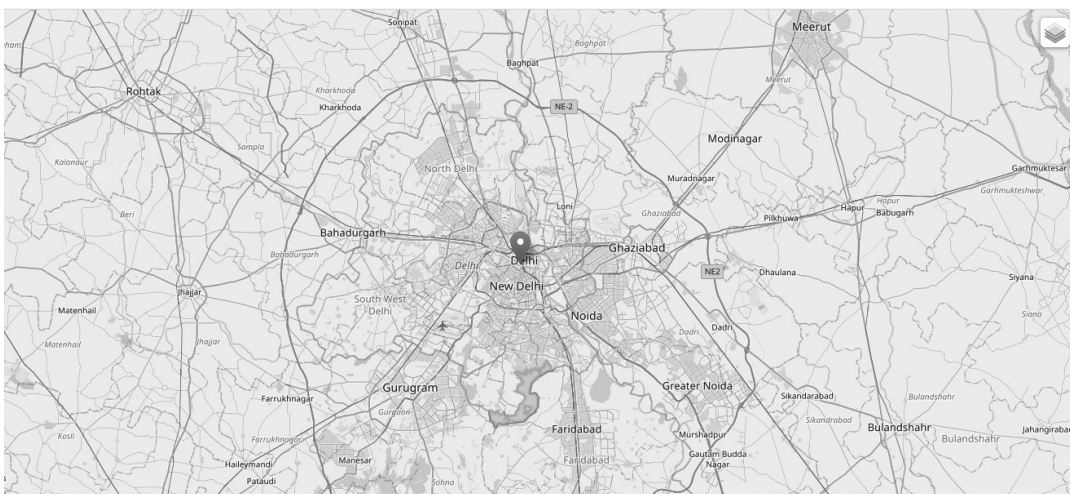
Adding Layers, Tiles and Markups on the Map

Tileset is a collection of raster and vector data presented using a uniform grid of square tiles. Folium plots maps with different tiles like Stamen Terrain, Stamen Toner, Stamen Water Color, CartoDB Positron, etc. The default tiles are set to OpenStreetMap. Each `tileset` represents different features of a map and is used for a specific purpose. For example, Stamen Terrain depicts hill shading and natural vegetation colors; whereas CartoDB Dark Matter shows the CartoDB Positron map in dark mode.

The biggest advantage of folium is that we can add different layers one over the other to plot more information on a single map. In the code given below, we have added five different tile layers to a single map. `LayerControl()` function provides an icon on the top-right corner of the map for switching between the different layers.

Markers on a map are used for marking a location on a map. For example, in Google Maps, your current location and destination location is marked by a marker. We use `folium.Marker()` class for plotting markers on a map. The function takes the latitude and longitude of the location. We can also add a popup and tooltip to the map.

```
import folium
import pandas as pd
from branca.element import Figure
fig2=Figure(width=550,height=350)
m2=folium.Map(location=[28.644800, 77.216721])
fig2.add_child(m2)
folium.TileLayer('Stamen Terrain').add_to(m2)
folium.TileLayer('Stamen Toner').add_to(m2)
folium.TileLayer('Stamen Water Color').add_to(m2)
folium.TileLayer('cartodbpositron').add_to(m2)
folium.TileLayer('cartodbdark_matter').add_to(m2)
folium.LayerControl().add_to(m2)
folium.Marker(location=[28.645800, 77.214721],popup='<strong>Marker3</strong>',tooltip='<strong>Click here to see Popup</strong>').add_to(m2)
# save map to html file
m2.save('index2.html')
```



Copyright © 2023 by McGraw Hill Education (India) Private Limited

A.34 Data Science and Machine Learning Using Python

Note that we can use the `folium.Icon()` class for creating custom icons for markers. `Icon()` takes three arguments – color, prefix and icon. To change shape of the marker, `folium.features.CustomIcon()` class is used. This function takes the path of the image and icon size in pixels as arguments and creates an icon object. This icon object can be passed to the `folium.Icon()` class as an icon for creating custom markers. For example, we can write the following statement to create a custom markup.

```
folium.Marker(location=[28.4411091,77.1167361],popup='Custom Marker  
2',tooltip='<strong>Click here to see Popup</strong>',icon=folium.Icon(color='green',prefix='glyphicon',icon='off')).add_to(m2)
```



Sentiment Analysis

Sentiment analysis falls under the category of text classification in which a phrase, or a list of phrases is given as input and the classifier specifies whether the sentiment behind that is positive, negative or neutral (Figure 1). For example, in the phrases given below,

1. "Dangal is a great movie."
2. "Dangal is not a great movie."
3. "Dangal is a movie."

Every phrase here, represents a different sentiment. The first phrase denotes positive sentiment, the second one is a negative sentiment and the third is a neutral one as there is no word that tells us how the movie is. Considering it a supervised learning task, we will pass phrases as input to the machine learning model, and test the model on unlabelled phrases.

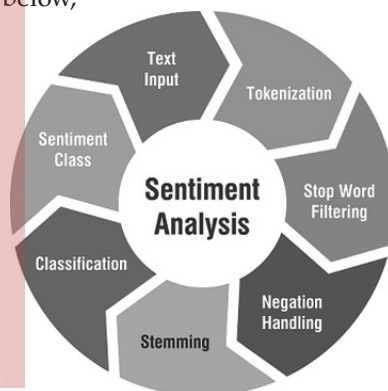


Figure 1 Uses of Sentiment Analysis

Importance of sentiment analysis

Sentiment analysis techniques are widely used to:

- predict customer behavior for a particular product.
- test the adaptability of a product.
- automate the generation of customer preference reports.
- determine how well is a movie or a product by analyzing its user's sentiments from a number of platforms.

Formulating the problem statement of sentiment analysis

To formally define the classification problem, we will use the following terminology.

Input:

- A document d
- A fixed set of classes $C = \{c_1, c_2, \dots, c_n\}$

Output: A predicted class $c \in C$

Here, *document* can be tweets, phrases, news articles or any other article, a product manual, a story, etc. So, a training set of n labeled documents looks like: $(d_1, c_1), (d_2, c_2), \dots, (d_n, c_n)$ and the ultimate output is a learned classifier.

Note that in sentiment analysis, not all the words in a phrase convey the sentiment of the phrase. Words like “I”, “Are”, “Am”, etc. or any other stop word do not reflect any sentiments and therefore useless in context of sentiment classification. So, here we need to do feature selection to identify the most relevant features that relate the most to the *class label*. Thus, a bag-of-words is formed. In the bag-of-words representation of a document, most relevant words and their frequencies of occurrences are specified. Mathematically, a bag is a set; so by the definition of a set, the bag contains only unique words.

Words in the bag-of-words are used to construct the feature set of the document. For example, if we have several documents containing movie reviews, then a bag-of-words is created for each one of them and their labels as positive, negative or neutral are added. The training data set looks like,

document	w1	w2	w3	w4	...	wn	sentiment
d1	2	1	3	1	...	1	positive
d2	1	5	5	5		1	negative
d3	3	8	6	8		2	positive
d4	2	5	1	5		3	positive
d5	3	0	3	0		0	negative
d6	0	0	0	0		0	negative
d7	2	0	0	0		0	positive
d8	9	2	9	2		2	negative
...

This representation is also known as **Corpus**. In the corpus, each row contains independent feature vectors (unique words) and their sentiments. All the features $w_1, w_2, w_3, w_4, \dots, w_n$ are generated from a bag of words. Note that it is not necessary that all the documents will contain each of these features/words.

A simple sentiment classifier in Python

In the example given below, we have used an offline movie review corpus. To use this, you must write `nlTK.download('all')`.

We will implement Naïve Bayes classifier on the movie reviews dataset that can be imported by writing, from `nlTK.corpus import movie_reviews`.

A list of documents is created and labeled with the appropriate categories.

For this, a feature extractor is formed. To limit the number of features that the classifier needs to process, a list of 2000 most frequent words is created in the corpus. We create a set of words because each word should be unique and checking whether a word occurs in a set is much faster than checking it in a list.

After extracting features, Naïve Bayes classifier is used to predict the sentiments of new movie reviews. We can also check classifier's performance by computing its accuracy on the test set. The two main advantages of Naïve Bayes classifier are reduced number of parameters and linear time complexity.


```

import nltk
#nltk.download('all')
from nltk.corpus import movie_reviews
import random
documents = [(list(movie_reviews.words(fileid)), category)
              for category in movie_reviews.categories()
              for fileid in movie_reviews.fileids(category)]
random.shuffle(documents)
# Define the feature extractor
all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
word_features = list(all_words)[:2000]
def document_features(document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in document_words)
    return features
# Train Naive Bayes classifier
featuresets = [(document_features(d), c) for (d,c) in documents]
train_set, test_set = featuresets[100:], featuresets[:100]
classifier = nltk.NaiveBayesClassifier.train(train_set)
# Test the classifier
print(nltk.classify.accuracy(classifier, test_set))
# Show the most important features as interpreted by Naive Bayes
classifier.show_most_informative_features(5)
# Train Naive Bayes classifier
featuresets = [(document_features(d), c) for (d,c) in documents]
train_set, test_set = featuresets[100:], featuresets[:100]
classifier = nltk.NaiveBayesClassifier.train(train_set)
# Test the classifier
print(nltk.classify.accuracy(classifier, test_set))
# Show the most important features as interpreted by Naive Bayes
classifier.show_most_informative_features(5)

```

OUTPUT

0.86

Most Informative Features

contains(outstanding) = True	pos : neg	=	10.8 : 1.0
contains(mulan) = True	pos : neg	=	9.0 : 1.0
contains(seagal) = True	neg : pos	=	7.8 : 1.0
contains(wonderfully) = True	pos : neg	=	6.4 : 1.0
contains(damon) = True	pos : neg	=	6.3 : 1.0

0.86

Most Informative Features

contains(outstanding) = True	pos : neg	=	10.8 : 1.0
contains(mulan) = True	pos : neg	=	9.0 : 1.0
contains(seagal) = True	neg : pos	=	7.8 : 1.0
contains(wonderfully) = True	pos : neg	=	6.4 : 1.0
contains(damon) = True	pos : neg	=	6.3 : 1.0

Let us consider another dataset containing tweets about six US airlines, and predict whether a tweet contains positive, negative, or neutral sentiment about the airline. We will use machine learning algorithms to train and test the sentiment analysis models.

For this, we will first import the dataset and observe the distribution of sentiments across all the tweets. It is clearly evident from the output that for almost all the airlines, majority of the tweets is negative, followed by neutral and positive tweets.

On closely analyzing the dataset, we see that feature set will consist of just the tweets that are present in the 11th column. The label set contains sentiment of the tweet (present in the second column) that we have to predict.

However, before building our sentiment analysis model, we have to clean the tweets as they may contain many slang words and punctuation marks. After the tweets are cleaned, we can divide the data set into two sets – training set and testing set.

Since statistical algorithms use mathematics to train machine learning models, we need to convert text to numbers. For this, we can use Bag of Words and TF-IDF scheme.

Since, we have set `max_features = 2500`, our algorithm will look for 2500 most frequently occurring words to create a bag of words feature vector. `max_df` specifies that only words that occur in a maximum of 80% of the documents are used. (Note that words present in all documents are too common to be used for classification). Similarly, `min_df = 7` means that the word should be present in at least 7 documents.

After converting the text into numbers, we divide our data into training and testing sets. The training set is used to train the algorithm and the testing set is used to evaluate the performance of the machine learning model.

Once the model has been trained, it is used to perform prediction using Random Forest algorithm. Finally, accuracy of the model is computed using confusion matrix.

```
import numpy as np
import pandas as pd
import re
import nltk
import matplotlib.pyplot as plt
data_source_url = "https://raw.githubusercontent.com/kolaveridi/kaggle-Twitter-US-Airline-Sentiment-/master/Tweets.csv"
airline_tweets = pd.read_csv(data_source_url)
print(airline_tweets.head())
import seaborn as sns
sns.barplot(x='airline_sentiment', y='airline_sentiment_confidence' ,
data=airline_tweets)
plt.show()
features = airline_tweets.iloc[:, 10].values
labels = airline_tweets.iloc[:, 1].values
processed_features = []
for sentence in range(0, len(features)):
    # Remove all the special characters
    processed_feature = re.sub(r'\W', ' ', str(features[sentence]))
    # remove all single characters
    processed_feature= re.sub(r'\s+[a-zA-Z]\s+', ' ', processed_feature)
    # Remove single characters from the start
```



```

processed_feature = re.sub(r'\^[a-zA-Z]\s+', ' ', processed_feature)
# Substituting multiple spaces with single space
processed_feature = re.sub(r'\s+', ' ', processed_feature, flags=re.I)
# Removing prefixed 'b'
processed_feature = re.sub(r'^b\s+', '', processed_feature)
# Converting to Lowercase
processed_feature = processed_feature.lower()
processed_features.append(processed_feature)
print(processed_features)
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer (max_features=2500, min_df=7, max_df=0.8,
stop_words=stopwords.words('english'))
processed_features = vectorizer.fit_transform(processed_features).toarray()
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(processed_features,
labels, test_size=0.2, random_state=0)
from sklearn.ensemble import RandomForestClassifier
text_classifier = RandomForestClassifier(n_estimators=200, random_state=0)
text_classifier.fit(X_train, y_train)
predictions = text_classifier.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
print(accuracy_score(y_test,predictions))

```

```

      tweet_id  ...  user_timezone
0  570306133677760513  ...  Eastern Time (US & Canada)
1  570301130888122368  ...  Pacific Time (US & Canada)
2  570301083672813571  ...  Central Time (US & Canada)
3  570301031407624196  ...  Pacific Time (US & Canada)
4  570300817074462722  ...  Pacific Time (US & Canada)
[5 rows x 15 columns]
[[1723  108   39]
 [ 326  248   40]
 [ 132   58 254]]

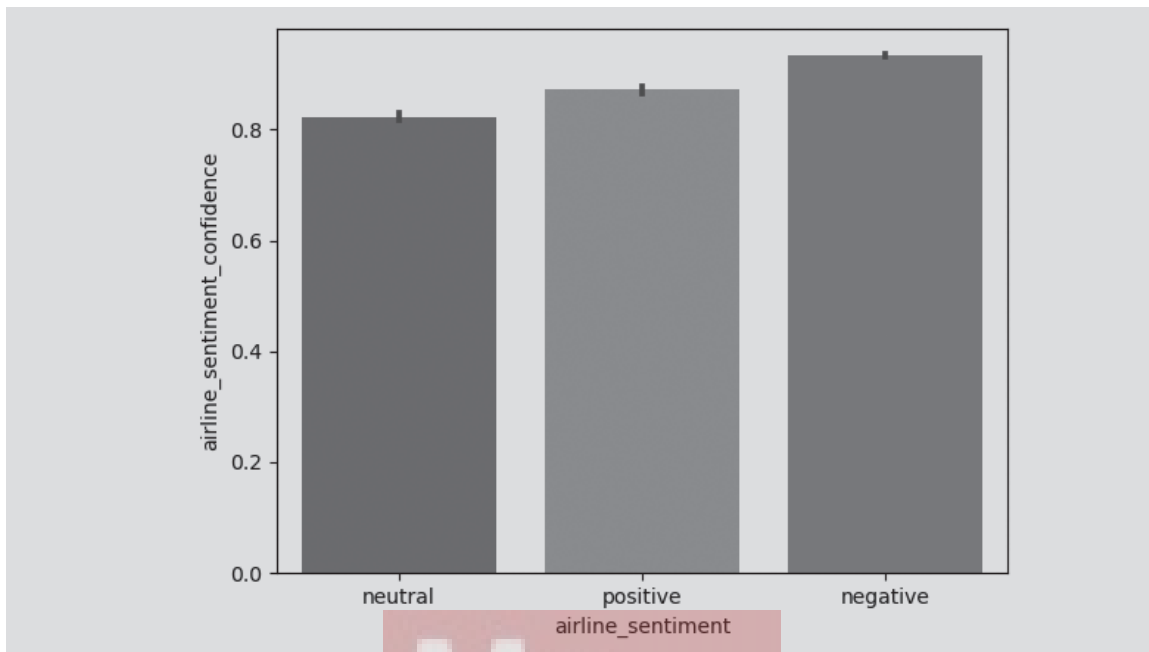
```

	precision	recall	f1-score	support
negative	0.79	0.92	0.85	1870
neutral	0.60	0.40	0.48	614
positive	0.76	0.57	0.65	444
accuracy			0.76	2928
macro avg	0.72	0.63	0.66	2928
weighted avg	0.75	0.76	0.74	2928

```

0.7599043715846995

```

So, we got an accuracy of almost 76%. However, Naïve Bayes has one major limitation – it assumes presence of independent predictors. In real life, it is almost impossible to get an entirely independent set of predictors.