Page No.
Date

EXPERIMENT-1

AIM - To implement naive bayes classifier on a dataset (Diabetes dataset)

SOFTWARE USED - In machine learning, naive bayes classifier are a family of simple "probalistic classifier" based on applying bayes theorem with strong (naive) independence assumption between the features.

Naive bayes classifier are highly scalable requiring a number of parameter linear in the number of variables in a learning problem.

In statistics and computer scene, Bayer models are known under a variety of names but naive bayes is not necessarily a Bayesian method

Applications of Naive Bayes

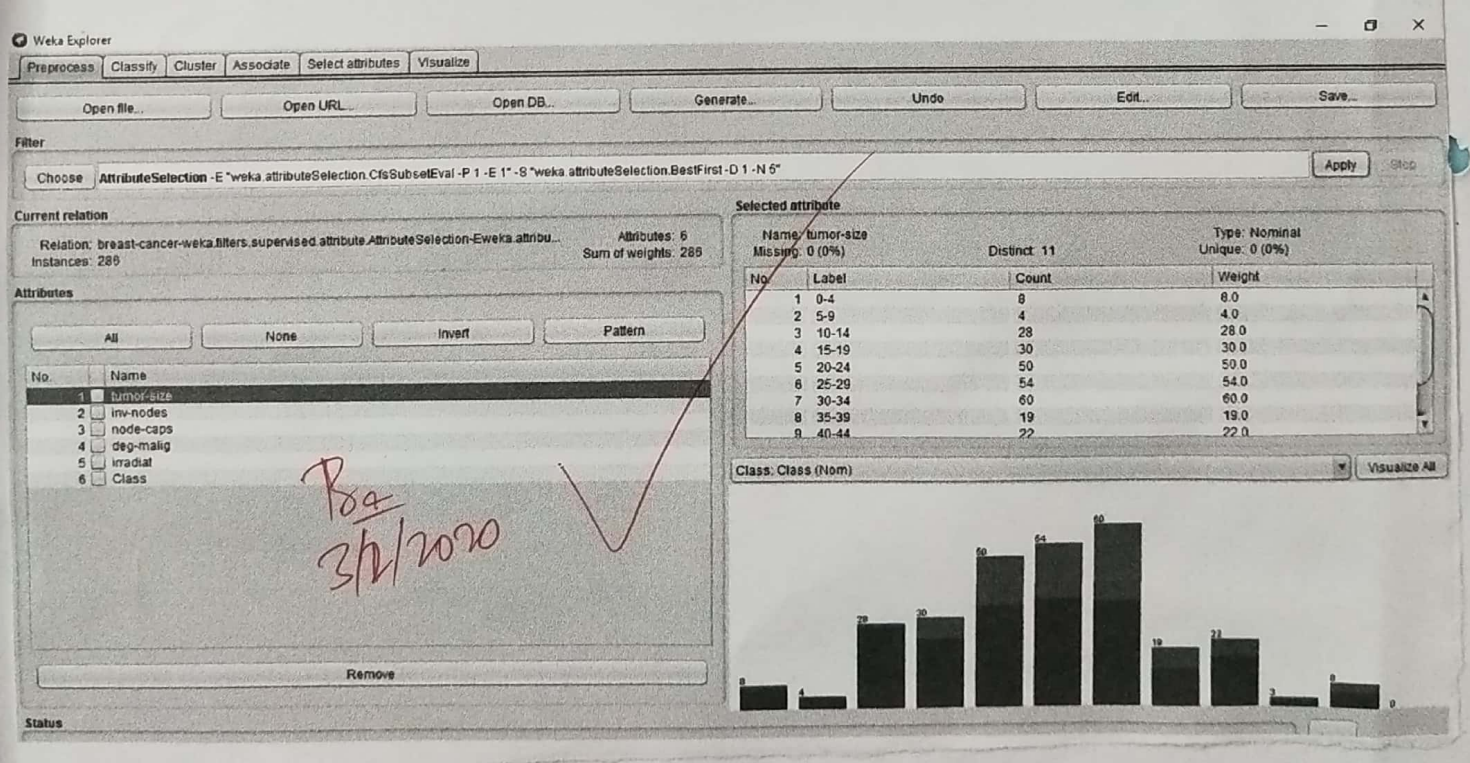
- Recommendation system
- Text classification / Sentiment analysis

RESULT: Naive Bayes was implemented successfully

Classifier output

```
Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         212               74.1259 %
Incorrectly Classified Instances        74               25.8741 %
Kappa statistic                          0.3721
Mean absolute error                      0.2889
Root mean squared error                  0.487
Relative absolute error                 69.0379 %
Root relative squared error            106.5497 %
Total Number of Instances              286

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   MCC      ROC Area  PRC Area  Class
                 0.826    0.459    0.810      0.826    0.818      0.372    0.703     0.807     no-recurrence-events
                 0.541    0.174    0.568      0.541    0.554      0.372    0.703     0.564     recurrence-events
Weighted Avg.    0.741    0.374    0.738      0.741    0.739      0.372    0.703     0.735

=== Confusion Matrix ===

   a   b   <-- classified as
 166  35 |   a = no-recurrence-events
  39  46 |   b = recurrence-events
```

Ra 3/12/2020



Ra 3/12/2020

Page No.
Date

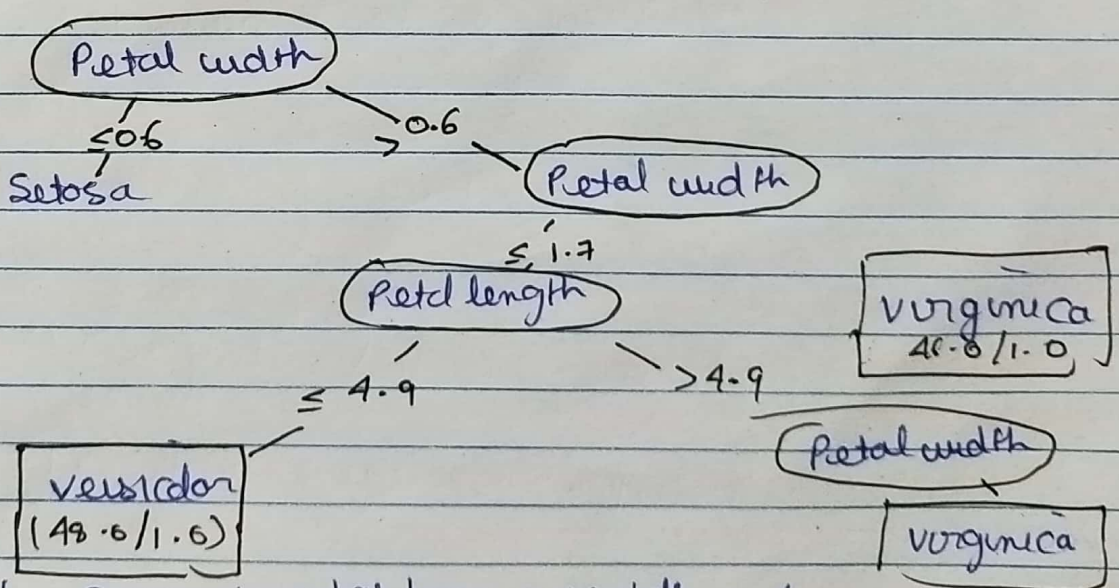
EXPERIMENT-2

AIM- To implement Decision tree algorithm on breast cancer dataset (J48 classifier)

SOFTWARE USED - Weka

THEORY - J48 has the full name weka classifier tree J48 C4.5 is an algorithm used to generate decision trees developed by Ross Quinlan

Imagine that you have a dataset with a list of predictors or dependent variables. Then by applying a decision tree like J48 on that dataset would allow you to predict the target value of new dataset record.



Result - Decision tree has been successfully used.

OBSERVATIONS :-

Classifier output

```
Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         216               75.5245 %
Incorrectly Classified Instances        70               24.4755 %
Kappa statistic                          0.2826
Mean absolute error                      0.3676
Root mean squared error                  0.4324
Relative absolute error                 87.8635 %
Root relative squared error             94.6093 %
Total Number of Instances              286

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
                 0.960    0.729    0.757      0.960   0.846      0.339    0.584     0.736     no-recurrence-events
                 0.271    0.040    0.742      0.271   0.397      0.339    0.584     0.436     recurrence-events
Weighted Avg.    0.755    0.524    0.752      0.755   0.713      0.339    0.584     0.647

=== Confusion Matrix ===

   a   b   <-- classified as
 193   8 |   a = no-recurrence-events
  62  23 |   b = recurrence-events
```

Ra
3/2/2020.

EXPERIMENT-3

AIM - Estimate the accuracy of decision classifier on breast cancer dataset using 5 fold cross validation

SOFTWARE USED -: WEKA

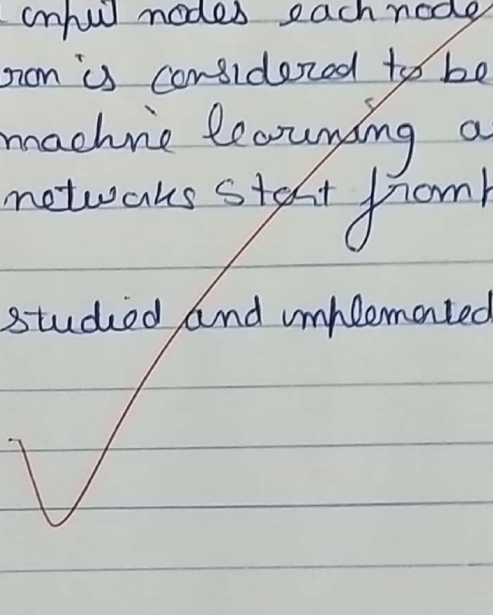
THEORY - Cross Validation is a model validation technique for assessing how the results of a statistical analysis will generalize an independent dataset. It is mainly used in settings where the goal is prediction and one wants to estimate how accurately a predictive model will perform in practice.
It is helpful because :

- It helps us evaluate the quality of model
- It selects the model which will perform best on unseen data
- Avoids underfitting and overfitting.

MULTI LAYER PERCEPTRON - is a class of feedforward artificial neural network (ANN). The term MLP refers to networks composed of multiple layers of perceptrons. An MLP consists of atleast 3 layers of

nodes. An input layer, a hidden layer and an output layer. Except for input nodes each node. The multilayer perceptron is considered to be the turning point in machine learning as the concept of neural networks start from here.

RESULT - MLP was studied and implemented sucessfully.



MULTI LAYER PERCEPTRON

Choose **MultilayerPerceptron** -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a

Test options

- Use training set
- Supplied test set — Set...
- Cross-validation Folds 5
- Percentage split % 66

More options...

(Nom) Class

Start | Stop

Result list (right-click for options)

- 00:59:08 - functions.MultilayerPerceptron
- 01:00:04 - functions.MultilayerPerceptron

Classifier output

```
Time taken to build model: 1.56 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         203               70.979 %
Incorrectly Classified Instances        83               29.021 %
Kappa statistic                          0.2352
Mean absolute error                      0.3348
Root mean squared error                  0.4782
Relative absolute error                 80.0051 %
Root relative squared error            104.6326 %
Total Number of Instances              286

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
                 0.861    0.647    0.759      0.861   0.807      0.243    0.643     0.768     no-recurrence-events
                 0.353    0.139    0.517      0.353   0.420      0.243    0.643     0.473     recurrence-events
Weighted Avg.    0.710    0.496    0.687      0.710   0.692      0.243    0.643     0.680

=== Confusion Matrix ===

   a   b   <-- classified as
 173  28 |   a = no-recurrence-events
  55  30 |   b = recurrence-events
```

EXPERIMENT-4

AIM- Estimate precision recall accuracy of decision tree classifier on text classification for each of 10 categories of 10 fold cross validation

THEORY- Cross validation is a model validation technique for assessing how results of a statistical analysis will generalize an independent dataset. It is mainly used in settings where the goal is prediction and one wants to estimate how accurately a predictive model will perform.

J48 is the full name weka classifier C4.5 is an an algorithm used to generate decision trees by Ross Quinlan.

- It helps us evaluate the quality of model
- Avoids underfitting and overfitting.

RESULT- The model was implemented successfully.

EXPERIMENT-4

=== Summary ===

Correctly Classified Instances	2948	54.7955 %
Incorrectly Classified Instances	2432	45.2045 %
Kappa statistic	0	
Mean absolute error	0.1847	
Root mean squared error	0.3039	
Relative absolute error	99.953 %	
Root relative squared error	100 %	
Total Number of Instances	5380	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1.000	1.000	0.548	1.000	0.708	0.000	0.499	0.548	Neg-
	0.000	0.000	0.000	0.000	0.000	0.000	0.498	0.042	Pos-Hyperplasia
	0.000	0.000	0.000	0.000	0.000	0.000	0.487	0.015	Pos-Mitosis
	0.000	0.000	0.000	0.000	0.000	0.000	0.497	0.077	Pos-Necrosis
	0.000	0.000	0.000	0.000	0.000	0.000	0.489	0.021	Pos-Pediatrics
	0.000	0.000	0.000	0.000	0.000	0.000	0.500	0.125	Pos-Pregnancy
	0.000	0.000	0.000	0.000	0.000	0.000	0.500	0.171	Pos-Rats
Weighted Avg.	0.548	0.548	0.300	0.548	0.388	0.000	0.499	0.353	

=== Confusion Matrix ===

a	b	c	d	e	f	g	<-- classified as
2948	0	0	0	0	0	0	a = Neg-
229	0	0	0	0	0	0	b = Pos-Hyperplasia
83	0	0	0	0	0	0	c = Pos-Mitosis
414	0	0	0	0	0	0	d = Pos-Necrosis
116	0	0	0	0	0	0	e = Pos-Pediatrics
670	0	0	0	0	0	0	f = Pos-Pregnancy
920	0	0	0	0	0	0	g = Pos-Rats

Page No.
Date

EXPERIMENT - 05

AIM - Develop a machine learning method to classify your incoming mail.

SOFTWARE USED - Jupyter Notebook / Python 3

THEORY - We have used here the UCIML SMS SPAM COLLECTION DATASET for spam classification. It has 5574 instances with each instance being a message that may be classified as spam.

EMAIL SPAM DETECTION - The spam filtering process can be considered as a classification problem from machine learning perspective. So we need to classify the message as spam or not spam depending upon its features.

NAIVE BAYES - Naive Bayes is a popular statistical technique of email filtering. They typically use bag of words features to identify spam email, an approach commonly used in text classifiers. Naive bayes works by correlating the used tokens

Page No.
Date

spam, non spam email & then uses bayes theorem to calculate a probabality that an email is spam are not

$$P(\text{spam} \mid w_1, w_2, \cdots w_n) \propto P(\text{spam}) \cdot \prod_{i=1}^{n} P(w_i \mid \text{spam})$$

The probabability of the message that contains words (w_1, w_2, w_3) to be spam is proportional to the probabality to get the spam multiplied by a product of probabilities for every word in the message to belong to a spam message

P_spam - part of spam messages in our dataset
P_w_i_spam - the probability of a word to be found in spam
P_not_spam & P_w_i_non spam are similar

The techniques implemented are:-
Bag of words and TF-IDF
Bag of words - we find term frequency P(w) and P(w/spam)
TF-IDF stands for Term frequency Inverse Document Freqeuncy.

RESULT- SMS spam classifier was implemented.

Accuracy	SVM	-	97%
	NAIVE BAYES		98%

Machine Learning Lab - Experiment - 5

Spam Classification

This notebook illustrates classification of SMS as SPAM or NOT SPAM.

Naive Bayes is the most simplest and most common ML Algo used for Spam Classification

Naive Bayes classification is a simple probability algorithm based on the fact, that all features of the model are independent. In the context of the spam filter, we suppose, that every word in the message is independent of all other words and we count them with the ignorance of the context.

Installing Dependencies

In [2]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import csv
import pandas
import sklearn
import pickle
from wordcloud import WordCloud
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.tree import DecisionTreeClassifier
from sklearn.learning_curve import learning_curve
```

Preprocessing and Exploring the Dataset

Importing the Dataset spam.csv

In [23]:

```
data = pd.read_csv('data/spam.csv', encoding='latin-1')
data.head()
```

Out[23]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

Removing unwanted columns

In [24]:

```
data = data.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1)
data = data.rename(columns={"v2" : "text", "v1":"label"})
```

In [25]:

```
data[1990:2000]
```

Out[25]:

	label	text
1990	ham	HI DARLIN IVE JUST GOT BACK AND I HAD A REALLY...
1991	ham	No other Valentines huh? The proof is on your ...
1992	spam	Free tones Hope you enjoyed your new content. ...
1993	ham	Eh den sat u book e kb liao huh...
1994	ham	Have you been practising your curtsey?
1995	ham	Shall i come to get pickle
1996	ham	Lol boo I was hoping for a laugh
1997	ham	\YEH I AM DEF UP4 SOMETHING SAT
1998	ham	Well, I have to leave for my class babe ... Yo...
1999	ham	LMAO where's your fish memory when I need it?

In [26]:

```
data['label'].value_counts()
```

Out[26]:

```
ham     4825
spam     747
Name: label, dtype: int64
```

In [29]:

```
import nltk
#nltk.download("punkt")
import warnings
warnings.filterwarnings('ignore')
```

In [30]:

```
ham_words = ''
spam_words = ''
```

In [31]:

```
for val in data[data['label'] == 'spam'].text:
    text = val.lower()
    tokens = nltk.word_tokenize(text)
    for words in tokens:
        spam_words = spam_words + words + ' '

for val in data[data['label'] == 'ham'].text:
    text = val.lower()
    tokens = nltk.word_tokenize(text)
    for words in tokens:
        ham_words = ham_words + words + ' '
```

In [32]:

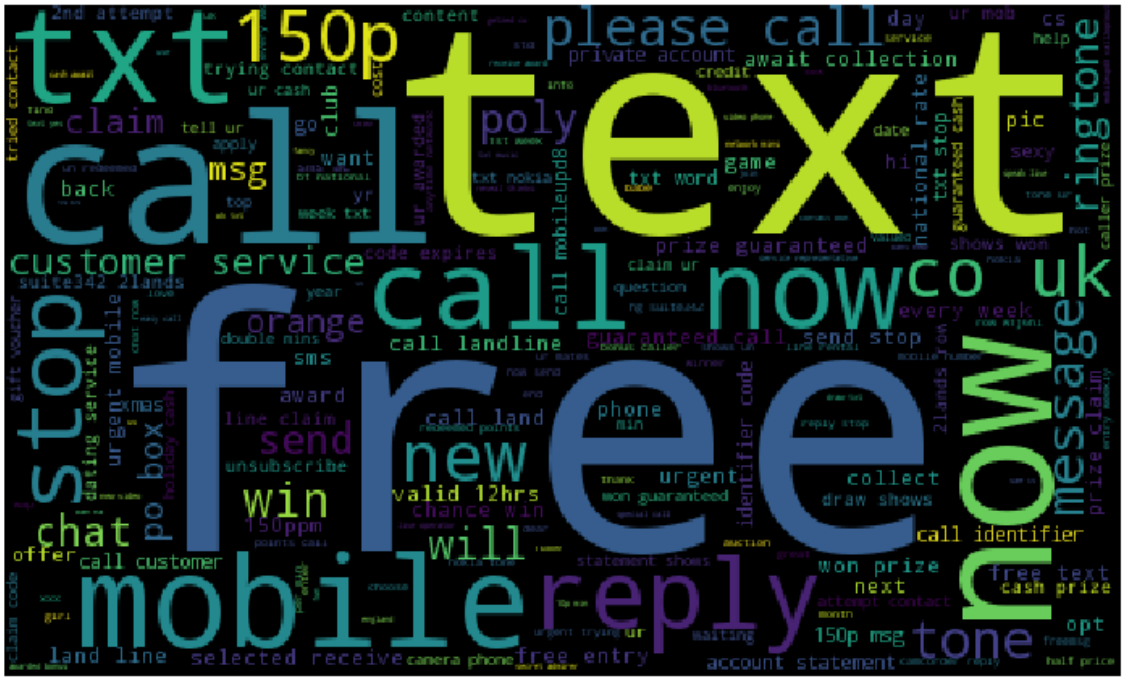
```
spam_wordcloud = WordCloud(width=500, height=300).generate(spam_words)
ham_wordcloud = WordCloud(width=500, height=300).generate(ham_words)
```

Creating a Word Cloud for Analysis of the Spam Words

Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance. Significant textual data points can be highlighted using a word cloud.

In [33]:

```
#Spam Word cloud
plt.figure( figsize=(10,8), facecolor='w')
plt.imshow(spam_wordcloud)
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()
```



In [35]:

```
data = data.replace(['ham','spam'],[0, 1])
```

In [37]:

```
data.head(10)
```

Out[37]:

	label	text	length
0	0	Go until jurong point, crazy.. Available only ...	111
1	0	Ok lar... Joking wif u oni...	29
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	0	U dun say so early hor... U c already then say...	49
4	0	Nah I don't think he goes to usf, he lives aro...	61
5	1	FreeMsg Hey there darling it's been 3 week's n...	148
6	0	Even my brother is not like to speak with me. ...	77
7	0	As per your request 'Melle Melle (Oru Minnamin...	160
8	1	WINNER!! As a valued network customer you have...	158
9	1	Had your mobile 11 months or more? U R entitle...	154

Removing Stopwords from the messages

In [41]:

```
import string
def text_process(text):

    text = text.translate(str.maketrans('', '', string.punctuation))
    text = [word for word in text.split() if word.lower() not in stopwords.words('engli

    return " ".join(text)
```

In [42]:

```
data['text'] = data['text'].apply(text_process)
```

In [44]:

```
data.head()
```

Out[44]:

	label	text	length
0	0	Go jurong point crazy Available bugis n great ...	111
1	0	Ok lar Joking wif u oni	29
2	1	Free entry 2 wkly comp win FA Cup final tkts 2...	155
3	0	U dun say early hor U c already say	49
4	0	Nah dont think goes usf lives around though	61

In [111]:

```
text = pd.DataFrame(data['text'])
label = pd.DataFrame(data['label'])
```

Converting words to vectors

- First create a vocabulary of all words in the dataset (text messages)
- **Vector created as follows :**
  - positions with respect to highest occuring word
  - Eg : 1 at first index means first word in vocab(most frequent occuring in vocab which is 'of') occurs twice in this sentence

In [112]:

```
## Counting how many times a word appears in the dataset

from collections import Counter

total_counts = Counter()
for i in range(len(text)):
    for word in text.values[i][0].split(" "):
        total_counts[word] += 1

print("Total words in data set: ", len(total_counts))
```

```
Total words in data set:  11305
```

In [113]:

```
## Sorting in decreasing order (Word with highest frequency appears first)
vocab = sorted(total_counts, key=total_counts.get, reverse=True)
print(vocab[:60])
```

```
['u', '2', 'call', 'U', 'get', 'Im', 'ur', '4', 'ltgt', 'know', 'go', 'lik
e', 'dont', 'come', 'got', 'time', 'day', 'want', 'Ill', 'lor', 'Call', 'hom
e', 'send', 'going', 'one', 'need', 'Ok', 'good', 'love', 'back', 'n', 'stil
l', 'text', 'im', 'later', 'see', 'da', 'ok', 'think', 'Ì', 'free', 'FREE',
'r', 'today', 'Sorry', 'week', 'phone', 'mobile', 'cant', 'tell', 'take', 'm
uch', 'night', 'way', 'Hey', 'reply', 'work', 'make', 'give', 'new']
```

In [114]:

```
# Mapping from words to index

vocab_size = len(vocab)
word2idx = {}
#print vocab_size
for i, word in enumerate(vocab):
    word2idx[word] = i
```

In [115]:

```
### Text to Vector
def text_to_vector(text):
    word_vector = np.zeros(vocab_size)
    for word in text.split(" "):
        if word2idx.get(word) is None:
            continue
        else:
            word_vector[word2idx.get(word)] += 1
    return np.array(word_vector)
```

In [117]:

```
## Convert all titles to vectors
word_vectors = np.zeros((len(text), len(vocab)), dtype=np.int_)
for ii, (_, text_) in enumerate(text.iterrows()):
    word_vectors[ii] = text_to_vector(text_[0])
```

In [118]:

```
word_vectors.shape
```

Out[118]:

```
(5572, 11305)
```

Converting words to vectors using TFIDF Vectorizer

In [156]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
vectors = vectorizer.fit_transform(data['text'])
vectors.shape
```

Out[156]:

```
(5572, 9376)
```

Choosing which algorithm we want to use a features : TFIDF or using custom vocabulary ?

In [157]:

```
#features = word_vectors
features = vectors
```

Splitting into training and test set

In [158]:

```
X_train, X_test, y_train, y_test = train_test_split(features, data['label'], test_size=
```

In [159]:

```
print (X_train.shape)
print (X_test.shape)
print (y_train.shape)
print (y_test.shape)
```

```
(4736, 9376)
(836, 9376)
(4736,)
(836,)
```

Classifying using sklearn pre built classifiers

In [163]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [164]:

```
svc = SVC(kernel='sigmoid', gamma=1.0)
knc = KNeighborsClassifier(n_neighbors=49)
mnb = MultinomialNB(alpha=0.2)
dtc = DecisionTreeClassifier(min_samples_split=7, random_state=111)
lrc = LogisticRegression(solver='liblinear', penalty='l1')
rfc = RandomForestClassifier(n_estimators=31, random_state=111)
```

In [165]:

```
clfs = {'SVC' : svc,'KN' : knc, 'NB': mnb, 'DT': dtc, 'LR': lrc, 'RF': rfc}
```

In [166]:

```
def train(clf, features, targets):
    clf.fit(features, targets)

def predict(clf, features):
    return (clf.predict(features))
```

In [167]:

```
pred_scores_word_vectors = []
for k,v in clfs.items():
    train(v, X_train, y_train)
    pred = predict(v, X_test)
    pred_scores_word_vectors.append((k, [accuracy_score(y_test , pred)]))
```

Predictions using TFIDF Vectorizer algorithm

In [168]:

```
predictions = pd.DataFrame.from_items(pred_scores,orient='index', columns=['Score'])
predictions
```

Out[168]:

	Score
SVC	0.978469
KN	0.933014
NB	0.988038
DT	0.960526
LR	0.953349
RF	0.979665

Predictions using custom vocabulary

In [132]:

```
predictions_word_vectors = pd.DataFrame.from_items(pred_scores_word_vectors,orient='inc
predictions_word_vectors
```

Out[132]:

	Score_
SVC	0.922249
KN	0.867225
NB	0.977273
DT	0.971292
LR	0.983254
RF	0.973684

Conclusion

By using various classifiers, we have shown that these ML algorithms can be easily used to classify data as spam or not spam. This can range from Spam SMS to Emails etc.

In [ ]:

```

```

Page No.
Date

EXPERIMENT - 06

AIM - Develop a machine learning algorithm to predict how people rate movies.

SOFTWARE USED - Python 3 - Jupyter Notebook

THEORY - We have to determine whether a given review is a positive or negative review. Thus we use Sentiment Analysis. It is a Natural Language Processing problem where text is understood and the underlying intent is predicted.

ML ALGORITHM APPLIED

RANDOM FOREST - Random forests are an ensemble machine learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean prediction of individual trees.

Page No.
Date

MULTINOMIAL NAIVE BAYES - Naive Bayes are a family of simple probabilistic classifiers based on applying bayes theorem with strong independence assumptions between the features.

Multinomial distribution is just a generalization of the binomial distribution derived using a couple combinatorics concepts. It simply assumes multinomial distribution for all the priors.

DATASET USED - We have used IMDB dataset. It contains 25,000 highly polar movie reviews (good or bad) for training and the same amount again for testing. The dataset was collected by stanford researchers in 2011.

CONCLUSION - The movie reviews were analyzed.

Multinomial Naive Bayes - 0.83308 on Test Set

Random Forest - 0.84712 on Test Set

Machine Learning Lab - Experiment - 6

Movie Review Prediction

The large movie view dataset contains a collection of 50,000 reviews from IMDB. The dataset contains an even number of positive and negative reviews. The dataset is divided into training and test sets. The training set is the same 25,000 labeled reviews.

We are going to use RandomForest and Multinomial Naive Bayes classifer for the prediction in this experiment.

In [1]:

```
import pandas as pd
import numpy as np
import glob, os, string, re, spacy
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report, accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB

```

Import datasets

In [2]:

```
train_pos_files = glob.glob("aclImdb/train/pos/*.txt")
train_neg_files = glob.glob("aclImdb/train/neg/*.txt")
train_pos_ls = []

for i in train_pos_files:
    file = open(i, "r")
    str = file.readline()
    clean = re.compile('<.*?>')
    str = re.sub(clean, ' ', str)
    train_pos_ls.append(str)

train_neg_ls = []
for i in train_neg_files:
    file = open(i, "r")
    str = file.readline()
    clean = re.compile('<.*?>')
    str = re.sub(clean, ' ', str)
    train_neg_ls.append(str)


```

In [3]:

```
labels = ['reveiw', 'label']
df_train_pos = pd.DataFrame()
df_train_pos['review'] = train_pos_ls
df_train_pos['label'] = 1
df_train_neg = pd.DataFrame()
df_train_neg['review'] = train_neg_ls
df_train_neg['label'] = -1
df_train = pd.concat([df_train_pos , df_train_neg])

```

In [4]:

```
test_pos_files = glob.glob("aclImdb/test/pos/*.txt")
test_neg_files = glob.glob("aclImdb/test/neg/*.txt")
test_pos_ls = []
for i in test_pos_files:
    file = open(i, "r")
    str = file.readline()
    clean = re.compile('<.*?>')
    str = re.sub(clean, ' ', str)
    test_pos_ls.append(str)

test_neg_ls = []
for i in test_neg_files:
    file = open(i, "r")
    str = file.readline()
    clean = re.compile('<.*?>')
    str = re.sub(clean, ' ', str)
    test_neg_ls.append(str)

```

In [5]:

```
labels = ['reveiw', 'label']
df_test_pos = pd.DataFrame()
df_test_pos['review'] = test_pos_ls
df_test_pos['label'] = 1
df_test_neg = pd.DataFrame()
df_test_neg['review'] = test_neg_ls
df_test_neg['label'] = -1
df_test = pd.concat([df_test_pos , df_test_neg])
df_test.head()
```

Out[5]:

	review	label
0	Based on an actual story, John Boorman shows t...	1
1	This is a gem. As a Film Four production - the...	1
2	I really like this show. It has drama, romance...	1
3	This is the best 3-D experience Disney has at ...	1
4	Of the Korean movies I've seen, only three had...	1

In [6]:

```
# Define text pre-processing functions
lemma = WordNetLemmatizer()
stops = set(stopwords.words('english'))

# nltk stopwords removal performs better than spacy
# nlp = spacy.load('en_core_web_sm')
# spacy_stopwords = spacy.lang.en.stop_words.STOP_WORDS

def text_prep(text):
    no_punct = [char for char in text if char not in string.punctuation]
    text = "".join(no_punct)
    text = [lemma.lemmatize(text, pos='v') for text in text.lower().split() if text not
    text = " ".join(text)
    return (text)

```

Data Preprocessing

In [7]:

```
df_train['prep_review'] = df_train['review'].apply(lambda x:text_prep(x))
df_train[['prep_review', 'label']].head()
```

Out[7]:

	prep_review	label
0	movie get respect sure lot memorable quote lis...	1
1	bizarre horror movie fill famous face steal cr...	1
2	solid unremarkable film matthau einstein wonde...	1
3	strange feel sit alone theater occupy parent r...	1
4	probably already know 5 additional episodes ne...	1

In [8]:

```
# preprocess testing data
df_test['prep_review'] = df_test['review'].apply(lambda x:text_prep(x))
df_test[['prep_review', 'label']].head()
```

Out[8]:

	prep_review	label
0	base actual story john boorman show struggle a...	1
1	gem film four production anticipate quality in...	1
2	really like show drama romance comedy roll one...	1
3	best 3d experience disney themeparks certainly...	1
4	korean movies ive see three really stick first...	1

In [9]:

```
# Vectorizing training data
tfidf = TfidfVectorizer()
# tfidf = TfidfVectorizer(ngram_range = (1,3)) did not improve accuracy
x_train = tfidf.fit_transform(df_train['prep_review'])
y_train = df_train['label']
```

In [10]:

```
# Vectorizing testing data
x_test = tfidf.transform(df_test['prep_review'])
y_test = df_test['label']
```

Prediction Models

Random Forest Classifier

In [20]:

```
RFC = RandomForestClassifier(n_estimators=100, random_state = 42, n_jobs = -1)
RFC.fit(x_train, y_train)
RFC_clf = RFC.predict(x_test)

```

In [21]:

```
RFC.score(x_train, y_train)
```

Out[21]:

```
1.0
```

In [22]:

```
accuracy_score(y_test, RFC_clf)
```

Out[22]:

```
0.84712
```

Multinomial Naive Bayes Classifier

In [23]:

```
MNB = MultinomialNB()
MNB.fit(x_train, y_train)
MNB_clf = MNB.predict(x_test)

```

In [24]:

```
1 MNB.score(x_train, y_train)
```

Out[24]:

```
0.9172
```

In [25]:

```
1 accuracy_score(y_test, MNB_clf)
```

Out[25]:

```
0.83308
```

In [ ]:

```
1
```

Page No.
Date

EXPERIMENT-07

AIM - Select two datasets. Each dataset should contain examples from multiple classes. For training purposes, assume that each class is unknown and implement KMeans algo and apply to data selected. Test the performance of algo as function of parameter K. Suggest a method for automatically determining number of clusters.

SOFTWARE USED - Jupyter notebook / Python3

THEORY - clustering is the process of dividing the entire data into groups (also known as clusters) based on the patterns in the data. It is an unsupervised learning problem.

KMeans Clustering - Kmeans clustering is one of the simplest unsupervised learning algorithm that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters fixed apriori. The main idea is to define k centers, each for one cluster.

Page No.
Date

STEPS - Let $X = \{x_1, x_2, x_3 \cdots x_n\}$ be set of data points.
$V = \{v_1, v_2 \cdots v_c\}$ be set of centers.

- Randomly select 'c' cluster centers
- Calculate the distance between each data point & cluster centers
- Assign datapoint to clustercenter whose distance from the cluster center is minimin of all cluster centers.
- Recalculate new cluste center using

$$v_i = (1/c_i) \sum_{j=1}^{c_i} x_i \quad \text{where } c_i = \text{no. data of points}$$

- Recalculate distance b/w each data point.

EVALUATION METRICS

INERTIA - It calculates sum of distances of all points within a cluster from centroid of cluster. The distance within clusters is called Intracluster distance. So inertia gives sum of cluster distance.

DUNN INDEX - $\min(\text{Interclusterdistance}) / \max(\text{Intraclusterdistance})$
It defines distance b/w 2 clusters

CONCLUSION - The Kmeans clustering was implemented successfully.

Machine Learning Lab - Experiment - 7

Using KMEANS CLUSTERING

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms. Typically, unsupervised algorithms make inferences from datasets using only input vectors without referring to known, or labelled, outcomes.

k-means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells. It is popular for cluster analysis in data mining.

Importing standard libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.cluster import KMeans
```

We are using the Wholesale Customer Segmentation Dataset here

We need to segment the clients of a wholesale distributor based on their annual spending on diverse product categories, like milk, grocery, region, etc.

In [3]:

```
data=pd.read_csv("Wholesale customers data.csv")
data.head()
```

Out[3]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

Statistics of the data

In [4]:

```
1 data.describe()
```

Out[4]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Deter
count	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	
mean	1.322727	2.543182	12000.297727	5796.265909	7951.277273	3071.931818	
std	0.468052	0.774272	12647.328865	7380.377175	9503.162829	4854.673333	
min	1.000000	1.000000	3.000000	55.000000	3.000000	25.000000	
25%	1.000000	2.000000	3127.750000	1533.000000	2153.000000	742.250000	
50%	1.000000	3.000000	8504.000000	3627.000000	4755.500000	1526.000000	
75%	2.000000	3.000000	16933.750000	7190.250000	10655.750000	3554.250000	
max	2.000000	3.000000	112151.000000	73498.000000	92780.000000	60869.000000	4

Scaling

In [5]:

```
1 from sklearn.preprocessing import StandardScaler
2 scaler = StandardScaler()
3 data_scaled = scaler.fit_transform(data)
4
5 # statistics of scaled data
6 pd.DataFrame(data_scaled).describe()
```

Out[5]:

	0	1	2	3	4	5	
count	4.400000e+02	4.400000e+02	4.400000e+02	4.400000e+02	4.400000e+02	4.400000e+02	4
mean	-2.452584e-16	-5.737834e-16	-2.422305e-17	-1.589638e-17	-6.030530e-17	1.135455e-17	
std	1.001138e+00	1.001138e+00	1.001138e+00	1.001138e+00	1.001138e+00	1.001138e+00	
min	-6.902971e-01	-1.995342e+00	-9.496831e-01	-7.787951e-01	-8.373344e-01	-6.283430e-01	
25%	-6.902971e-01	-7.023369e-01	-7.023339e-01	-5.783063e-01	-6.108364e-01	-4.804306e-01	-
50%	-6.902971e-01	5.906683e-01	-2.767602e-01	-2.942580e-01	-3.366684e-01	-3.188045e-01	
75%	1.448652e+00	5.906683e-01	3.905226e-01	1.890921e-01	2.849105e-01	9.946441e-02	
max	1.448652e+00	5.906683e-01	7.927738e+00	9.183650e+00	8.936528e+00	1.191900e+01	7

In [6]:

```
kmeans = KMeans(n_clusters=2, init='k-means++')

# fitting the k means algorithm on scaled data
kmeans.fit(data_scaled)
```

Out[6]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=2, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [7]:

```
# inertia on the fitted data
kmeans.inertia_
```

Out[7]:

```
2599.38555935614
```

Determining the number of clusters

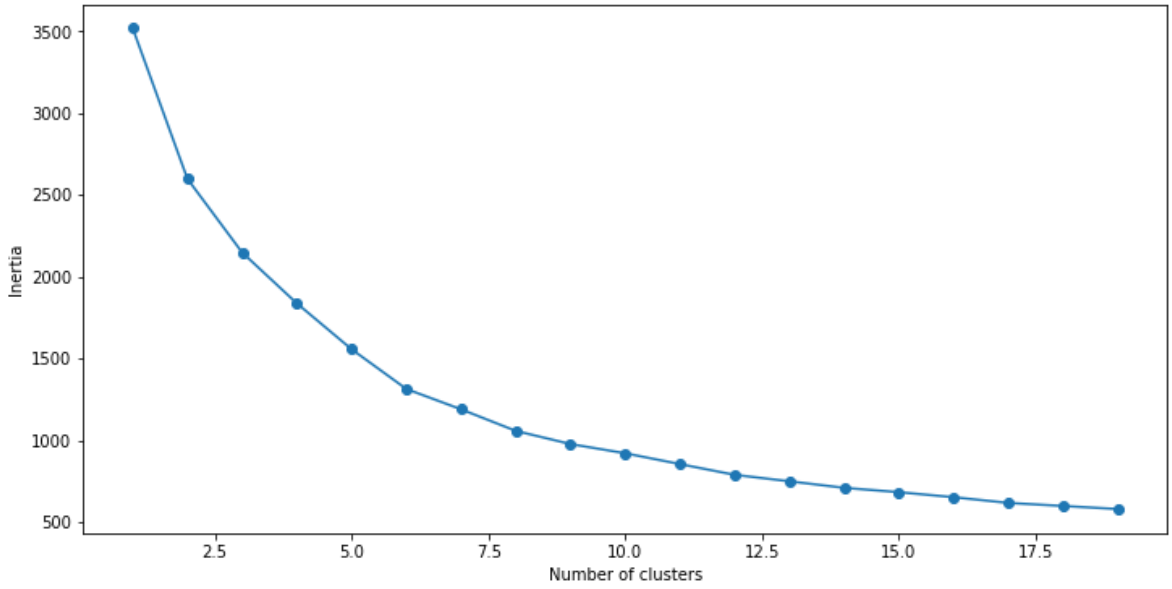
We will first fit multiple k-means models and in each successive model, we will increase the number of clusters. We will store the inertia value of each model and then plot it to visualize the result:

In [8]:

```
# fitting multiple k-means algorithms and storing the values in an empty list
SSE = []
for cluster in range(1,20):
    kmeans = KMeans(n_jobs = -1, n_clusters = cluster, init='k-means++')
    kmeans.fit(data_scaled)
    SSE.append(kmeans.inertia_)

# converting the results into a dataframe and plotting them
frame = pd.DataFrame({'Cluster':range(1,20), 'SSE':SSE})
plt.figure(figsize=(12,6))
plt.plot(frame['Cluster'], frame['SSE'], marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')

```

Out[8]:

```
Text(0,0.5,'Inertia')
```



Using Elbow Method

the basic idea behind partitioning methods, such as k-means clustering, is to define clusters such that the total intra-cluster variation [or total within-cluster sum of square (WSS)] is minimized. The total WSS measures the compactness of the clustering and we want it to be as small as possible.

The Elbow method looks at the total WSS as a function of the number of clusters: One should choose a number of clusters so that adding another cluster doesn't improve much better the total WSS.

Looking at the above elbow curve, we can choose any number of clusters between 5 to 8. The cluster value where this decrease in inertia value becomes constant can be chosen as the right cluster value for our data.

In [9]:

```
kmeans = KMeans(n_jobs = -1, n_clusters = 5, init='k-means++')
kmeans.fit(data_scaled)
pred = kmeans.predict(data_scaled)
```

In [10]:

```
frame = pd.DataFrame(data_scaled)
frame['cluster'] = pred
frame['cluster'].value_counts()
```

Out[10]:

```
1    208
0    125
4     91
2     10
3      6
Name: cluster, dtype: int64
```

Conclusion

The Elbow Curve method is helpful because it shows how increasing the number of the clusters contribute separating the clusters in a meaningful way, not in a marginal way.

The Elbow method is fairly clear, if not a naïve solution based on intra-cluster variance. The gap statistic method is more sophisticated method to deal with data that has a distribution with no obvious clustering (can find the correct number of k for globular, Gaussian-distributed, mildly disjoint data distributions).

In [ ]:

```

```

Page No.
Date

EXPERIMENT-8

AIM - To implement the EM algorithm assuming a Gaussian mixture. Apply the algorithm to your datasets and report the parameters you obtain. Evaluate the performance by measuring the sum of Manhabalis Distance of each example from its class center. Test the function as number of clusters.

SOFTWARE USED - Python 3 / Jupyter Notebook

THEORY - Clustering is concerned with grouping objects together that are similar to each other and dissimilar to objects belonging to other clusters. The EM (Expectation Maximization) algorithm extend this basic approach.

The EM clustering algorithm computes probability of cluster memberships based on one or more probability distributions. The goal of clustering algorithm is to maximize the overall probability or likelihood of data.

It estimates the mean & standard deviation for each cluster so as to maximize the likelihood of observed data (distribution)

Page No.
Date

A Gaussian Mixture Model uses an Expectation maximization approach

- Choose starting guess
- Repeat until converged
- E - step - for each point: find weights encoding the probability of membership
- M - step - In each cluster update its location normalization & shape.

MAHALANOBIS DISTANCE - It is the distance between a point and a distribution
It is a a multivariate equivalent of the euclidean distance. It was introduced by Prof PC Mahalanobis.

The formula to compute :

$$D^2 = (x-m)^T C^{-1} (x-m)$$

D^2 = Square of Mahalanobis
x - vector of observation
m - vector of mean values
C^{-1} inverse covariance matrix of independent variables

CONCLUSION - EM clustering was implemented successfully.

Machine Learning Lab - Experiment - 8 ¶

Implementing the EM Algorithm : assuming Gaussian distribution

The expectation-maximization algorithm is an approach for performing maximum likelihood estimation in the presence of latent variables. It does this by first estimating the values for the latent variables, then optimizing the model, then repeating these two steps until convergence. It is an effective and general approach and is most commonly used for density estimation with missing data, such as clustering algorithms like the Gaussian Mixture Model.

Importing standard libraries

In [1]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
```

In [2]:

```
import warnings
warnings.filterwarnings('ignore')
```

Generating some data

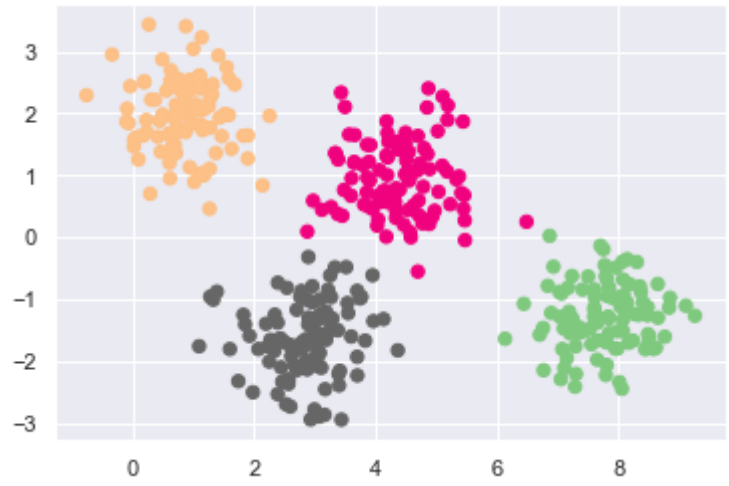
In [3]:

```
#generate some data
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples=400, centers=4,
                       cluster_std=0.60, random_state=0)
X = X[:, ::-1] # flip axes for better plotting
```

USING GMM

In [45]:

```
# from sklearn import mixture
# model = mixture.GaussianMixture(n_components=3, covariance_type='full')
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=4).fit(X)
labels = gmm.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='Accent');
```



GMM contains a probabilistic model under the hood, it is also possible to find probabilistic cluster assignments

In [7]:

```
probs = gmm.predict_proba(X)
```

In [8]:

```
probs
```

Out[8]:

```
array([[5.30761633e-01, 1.75162717e-22, 4.69238090e-01, 2.76240973e-07],
       [9.22826700e-10, 4.71110558e-15, 1.97106146e-17, 9.99999999e-01],
       [2.09565089e-09, 3.07981606e-17, 2.34875746e-14, 9.99999998e-01],
       ...,
       [4.50722543e-08, 9.99999933e-01, 2.32520153e-36, 2.15904343e-08],
       [5.36220515e-01, 3.80339560e-04, 2.87688722e-15, 4.63399146e-01],
       [1.19335720e-11, 1.00000000e+00, 1.10349655e-46, 6.20815080e-14]])
```

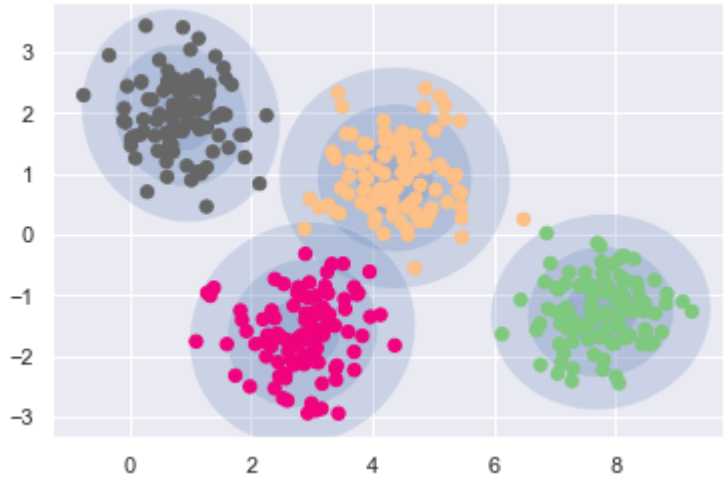
A function that will help us visualize the locations and shapes of the GMM clusters by drawing ellipses based on the GMM output:

In [40]:

```
from matplotlib.patches import Ellipse

def draw_ellipse(position, covariance, ax=None, **kwargs):
    """Draw an ellipse with a given position and covariance"""
    ax = ax or plt.gca()

    # Convert covariance to principal axes
    if covariance.shape == (2, 2):
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 2 * np.sqrt(covariance)

    # Draw the Ellipse
    for nsig in range(1, 4):
        ax.add_patch(Ellipse(position, nsig * width, nsig * height,
                             angle, **kwargs))

def plot_gmm(gmm, X, label=True, ax=None):
    ax = ax or plt.gca()
    labels = gmm.fit(X).predict(X)
    if label:
        ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='Accent', zorder=2)
    else:
        ax.scatter(X[:, 0], X[:, 1], s=40, zorder=2)
    ax.axis('equal')

    w_factor = 0.2 / gmm.weights_.max()
    for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):
        draw_ellipse(pos, covar, alpha=w * w_factor)
```

In [41]:

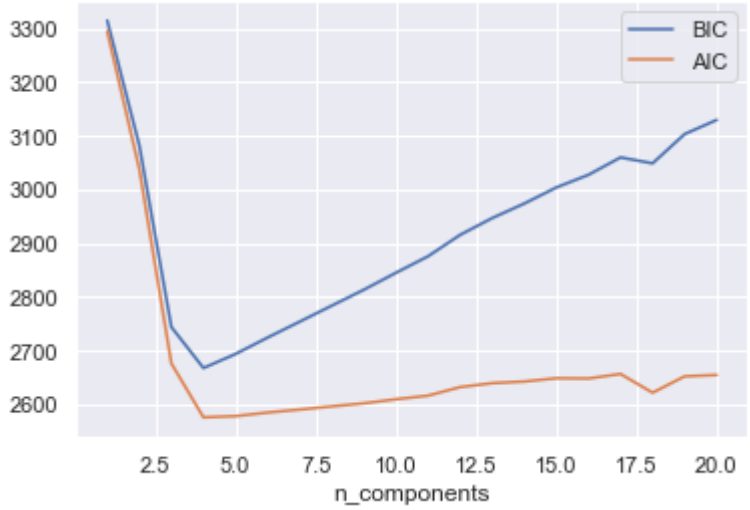
```
gmm = GaussianMixture(n_components=4, random_state=42)
plot_gmm(gmm, X)
```



OPTIMAL NUMBER OF CLUSTERS

In [16]:

```
n_components = np.arange(1, 21)
models = [GaussianMixture(n, covariance_type='full', random_state=0).fit(X)
          for n in n_components]

plt.plot(n_components, [m.bic(X) for m in models], label='BIC')
plt.plot(n_components, [m.aic(X) for m in models], label='AIC')
plt.legend(loc='best')
plt.xlabel('n_components');
```



The optimal number of clusters is the value that minimizes the AIC or BIC, depending on which approximation we wish to use.

Evaluating MAHALANOBIS DISTANCE

Mahalanobis distance is an effective multivariate distance metric that measures the distance between a point and a distribution. It is an extremely useful metric having, excellent applications in multivariate anomaly detection, classification on highly imbalanced datasets and one-class classification.

In [17]:

```
import pandas as pd
import scipy as sp
```

In [18]:

```
df = pd.DataFrame(X,index=X[:,0])
```

In [19]:

```
def mahalanobis(x=None, data=None, cov=None):
    """Compute the Mahalanobis Distance between each row of x and the data
    x    : vector or matrix of data with, say, p columns.
    data : ndarray of the distribution from which Mahalanobis distance of each observat
ion of x is to be computed.
    cov  : covariance matrix (p x p) of the distribution. If None, will be computed fro
m data.
    """
    x_minus_mu = x - np.mean(data)
    if not cov:
        cov = np.cov(data.values.T)
    inv_covmat = sp.linalg.inv(cov)
    left_term = np.dot(x_minus_mu, inv_covmat)
    mahal = np.dot(left_term, x_minus_mu.T)
    return mahal.diagonal()

df_x = df
df_x['mahala'] = mahalanobis(x=df_x,data=df)
df_x.head(10)
```

Out[19]:

	0	1	mahala
6.488267	6.488267	0.250064	1.555889
2.521947	2.521947	-2.697986	5.648692
3.068773	3.068773	-2.900076	5.545349
4.314275	4.314275	0.702951	0.372113
2.384499	2.384499	-2.535635	5.319597
7.805410	7.805410	-1.702978	2.308190
0.838773	0.838773	2.223222	2.211361
4.176143	4.176143	0.007931	0.010279
3.026685	3.026685	-1.790418	2.468821
1.269879	1.269879	1.767474	1.488402

Conclusion

The EM algorithm was implemented assuming a Gaussian Mixture. It is highly useful for class imbalanced dataset

In [ ]: