Gurvansh Singh
M.Tech
Knowledge Solutions India

Project 3:
NLP-Review classification

Team :
Mohammed Asra Fatima - Vidya Jyothi Institute of Technology
Ishika Gupta - Mahatma Gandhi Institute of Technology
Sabrina Shaik - Mahatma Gandhi Institute of Technology
Nadaf Razia Thabasum - Gates Institute of Technology

Table of content :

Index:

a. Table of figures /graphs :

Pictographic Representation:
Model 1: SVM

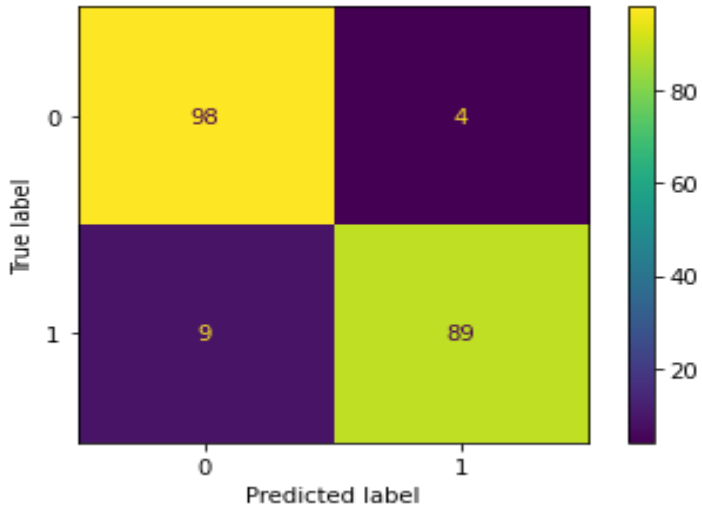


Figure 1a.

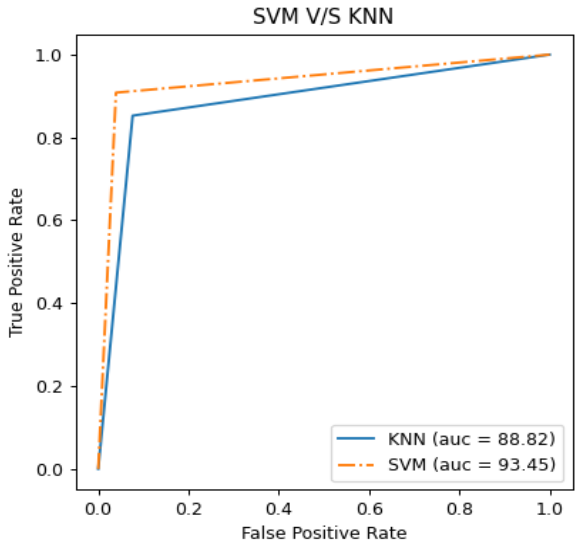


Figure 1b.

Model 2: KNN

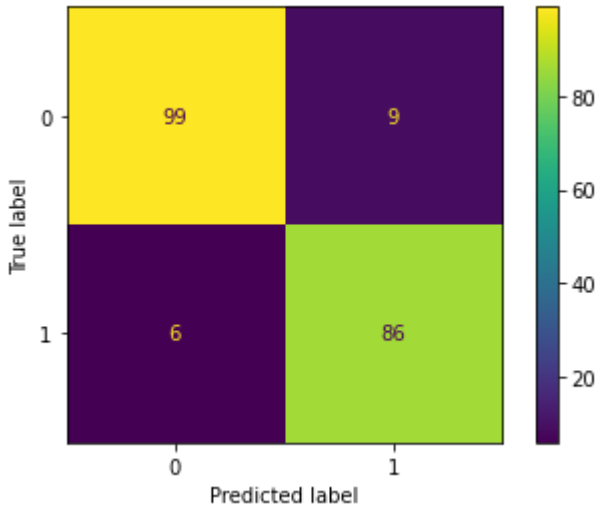


Figure 2a.

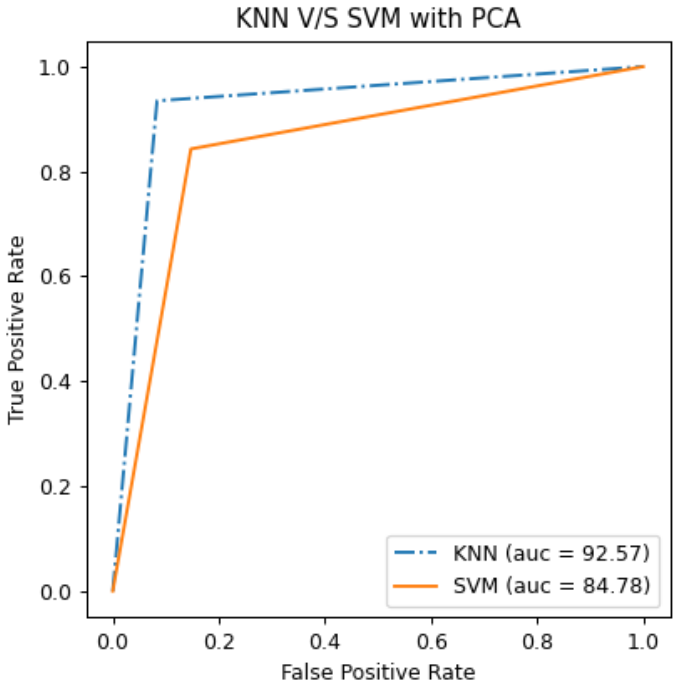


Figure 2b.

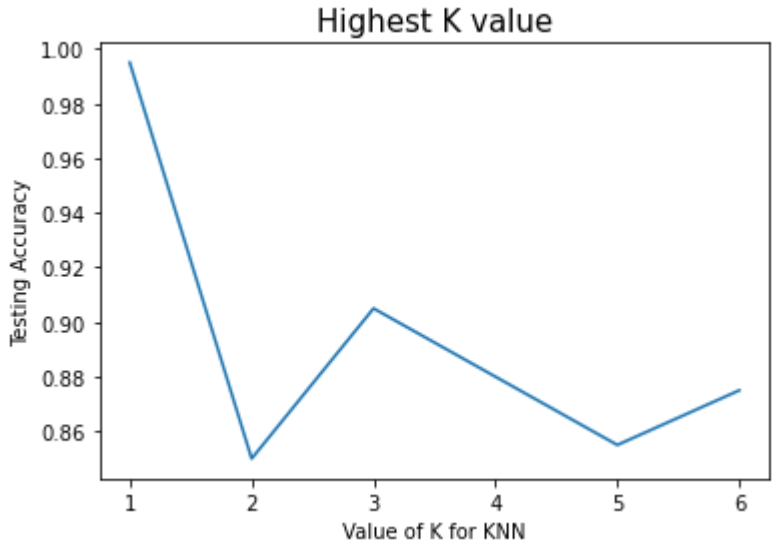


Figure 2c.

MODEL 3: Support Vector Machines with PCA

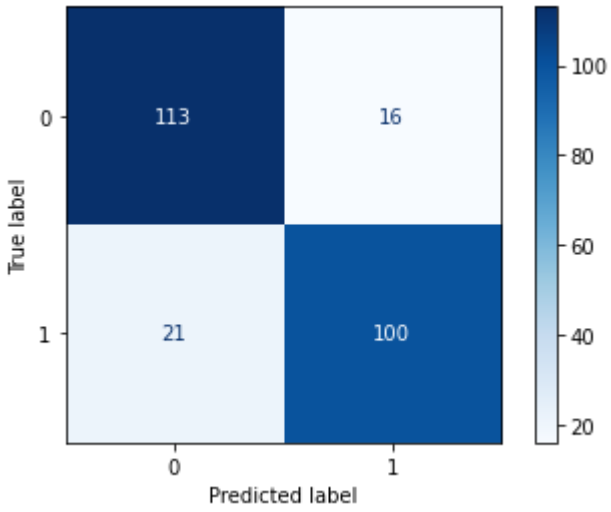


Figure 3a.

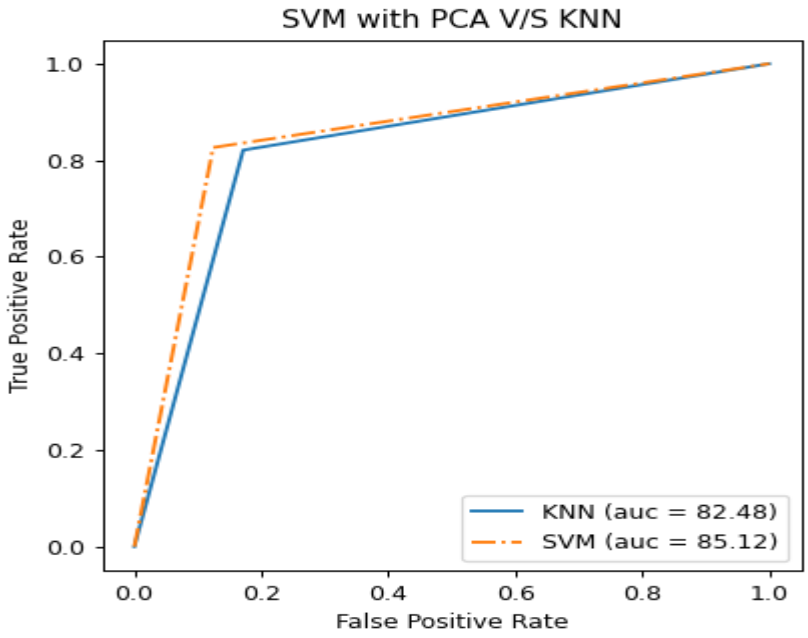
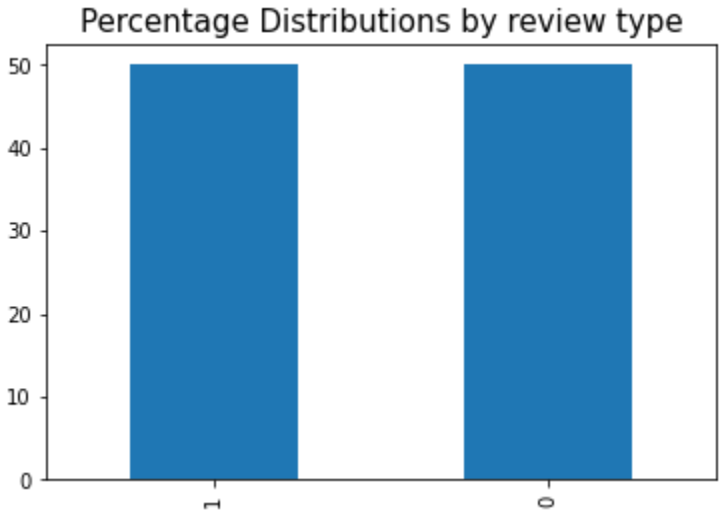


Figure 3b.

Visualizing the dataset , Negative and Positive reviews:




```
Percentage for default

1    50.0
0    50.0
Name: Liked, dtype: float64
```

c. Abstract:

The paper here presents a classification machine learning model to classify restaurant reviews. The reviews are either positive or negative based on the restaurant or the service it provides, the food of the restaurant, staff of the restaurant and the environment of the restaurant. A thorough study entailing what are the factors that influence the classification of their opinion on the restaurant has been taken into consideration with the help of various reviews given by the customers. Implementing machine learning algorithms we come to a conclusion on which classification algorithm works best with the given set of reviews.

The models used for classifying the reviews are:

1. **Support Vector Machine (SVM)**
2. **K-Nearest Neighbor (KNN)**
3. **Support Vector Machine with Principal Component Analysis (PCA)**

The classified reviews are helpful for the restaurant to analyze their shortcomings in different areas and improve the quality of food and service in the restaurant. The reviews are stored on cloud and can be accessed by whomsoever is concerned.

D. Chapters:

- **Introduction-**

In today's world technology and automation in every sector is rapidly increasing. People rely more on mobile devices for almost every task in their day to day lives. Restaurant Business is a sector which has a very large scope in automation and use of technology. At such emergent times waiting for the waiter to take orders, delivering food, lengthy queues, etc. can be very unpleasant for the customers of the restaurant. It is not guaranteed that the place they get their food from is authenticating delicious or not
To overcome these problems a concept of automated restaurants using a system which uses LCD displays, mobile/tablet devices to provide genuine reviews to the prospective customers is proposed. Using Machine Learning and data science predictions are made based on the reviews and other data of the customer can help make the dining experience better as well as it will help the restaurant to manage and make the restaurant grow.
In a restaurant while placing an order, the customer has to ask the waiter how the particular dish tastes. and even then there's a chance that the waiter could lie to enhance their expectations of the food, to get profits for the place he works at. To avoid this , customers can check online what are the reviews for the dish of their choice , in detail from its preparation to the price to be made available easily with a touch for just a few clicks. This is also convenient when the customer orders food online to be delivered at home , as sometimes we don't trust what happens behind the scenes of making the food.
Storing the statistical data of the restaurant is a very tedious task. There is need of managing the data of inventory, customer orders and reviews, staff and payroll. Therefore all this information is stored on cloud servers which are very efficient and cheap when compared to physical servers.

Software-Libraries used:

Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

The libraries used commonly for various tasks to perform are numpy, pandas and matplotlib.
Installation of Numpy in python: pip install numpy
Here Numpy imported "as" np provides a high-performance multidimensional array and basic tools to compute with and manipulate these arrays. SciPy builds on this, and provides a large number of functions that operate on numpy arrays and are useful for different types of scientific and engineering applications.
It also has functions for working in the domain of linear algebra, Fourier transform, and matrices.

Installation of Pandas in python: pip install pandas
Pandas imported as pd helps us in importing the dataset. It's key data structure is called the DataFrame. DataFrames allow you to store and manipulate tabular data in rows of observations and columns of variables.
The dataset used is a tsv file. A Tab Separated Values, which is imported as
"df = pd.read_csv('Restaurant_Reviews.tsv', delimiter='\t', quoting=3)"
Quoting = 3 denotes removing all the quotes in the data.
Quoting = 1 denotes removing single quotes
Quoting = 2 denotes removing double quotes

Installation of Matplotlib in python: pip install matplotlib
Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack.
One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Refining Data

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
corpus = [] # list will contain all the refined reviews
```

The above libraries are used for implementing Natural Language Processing.
(i) Importing re is a regularization library which helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern.

(ii) Importing nltk is the natural language toolkit which helps in tokenization , lemming .

(iii) The nltk.corpus package defines a collection of corpus reader classes, which can be used to access the contents of a diverse set of corpora.

(iv) We import stopwords to remove 'is', 'a', 'the', 'an', 'in' Stop words are commonly eliminated from many text processing applications because these words can be distracting, non-informative (or non-discriminative) and are additional memory overhead. Stop words are a set of commonly used words in any language.

Creating a bag of words

```
from sklearn.feature_extraction.text import CountVectorizer
import sklearn
cv = CountVectorizer(max_features=2500)
# using the max_feature parameter of countVectorizer to limit the number of columns in x
x = cv.fit_transform(corpus).toarray()
y = df.iloc[:,-1].values
len(x[0])
```

Installing sklearn in python: pip install -U scikit-learn
There will be thousands of words in the dataset. Some of the words will have a very small frequency. The words with very small frequency are not very useful, hence such words are removed.

They are removed using the CountVectorizer library which creates an array Of the most frequently used words in the dataset and fits this as the input to our model.

Feature Scaling:

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
sc = StandardScaler()
x = sc.fit_transform(x)
scaler = RobustScaler()
x = scaler.fit_transform(x)
```

Importing StandardScaler: Standardize features by removing the mean and scaling to unit variance. Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set.

Importing RobustScaler: Scale features using statistics that are robust to outliers. This Scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). In such cases, the median and the interquartile range often give better results.

Splitting the training and testing data :

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size = 0.75, random_state = 79)
```

Importing the train_test_split from sklearn helps us in extracting the training dataset as well as the testing dataset easily with a few parameters like the train/test size , the data to be split from(x,y) and the random_state

Applying Principal Component Analysis on SVM :

```
from sklearn.decomposition import PCA
pca = PCA(n_components= 2)
x = pca.fit_transform(x)
print("variance ratio: ", pca.explained_variance_ratio_)
```

```
variance ratio:  [0.02540401 0.02172652]
```

```
# kernel PCA
from sklearn.decomposition import KernelPCA

kpca = KernelPCA(n_components = 3, kernel = 'linear',tol=0.1,alpha=2.0) #n_components=3 kernel='linear'
x = kpca.fit_transform(x)
```

A common way of speeding up a machine learning algorithm is by using Principal Component Analysis (PCA). If the learning algorithm is too slow because the input dimension is too high, then using PCA to speed it up can be a reasonable choice. Another common application of PCA is for data visualization.

Creating and training the model for SVM and SVM with PCA:

```
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier
classifier= SVC(kernel='linear' , C=1000, gamma=1)
model = BaggingClassifier( base_estimator=classifier,
                           bootstrap=True,
                          n_estimators =100, # number of SVC models to create
                          max_samples=220,# each model is trained from randomly sampled 220 instances
                          random_state=49
                         )
model.fit(x_train, y_train)
```

Sklearn.svm is the package for support vector machines where we import SVC which is a support vector classifier. SVM is used for a subset of training points in the decision function (called support vectors), so it is also memory efficient.

Each base classifier is trained in parallel with a training set which is generated by randomly drawing, N examples(or data) from the original training dataset – where N is the size of the original training set. Training set for each of the base classifiers is independent of each other. Many of the original data may be repeated in the resulting training set while others may be left out. Bagging reduces overfitting by averaging or voting, however, this leads to an increase in bias, which is compensated by the reduction in variance.

Creating and training the KNN model:

```
#creating and training knn model
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=3,metric='minkowski',leaf_size=1, p=6)
classifier.fit(x,y)
```

Here , as we imported SVM earlier we import KNeighborsClassifier from sklearn.neighbors. In short , KNN provides us with various metrics like the one we used here 'minkowski', 'euclidean distance' , 'manhattan distance' and so on. N_neighbors is also one of the most important parameter for KNN , the higher the n_neighbors the higher is the computation time and lower the accuracy for the model.

Calculating Accuracy ,Precision and Recall using metrics library:

```
from sklearn import metrics
print("Accuracy:",format(metrics.accuracy_score(y_test, y_pred_bagging)*100,'.2f'))
print("Precision:",format(metrics.precision_score(y_test, y_pred_bagging)*100,'.2f'))
print("Recall:",format(metrics.recall_score(y_test, y_pred_bagging)*100,'.2f'))
```

Sklearn provides us with a wide range of tool kits, one of them being metrics which help us in estimating our accuracy ,precision and recall which is further used in AUC ROC curve.

Plotting the Confusion matrix using sklearn.metrics:

```
# plotting the confusion matrix and the accuracy score
from sklearn.metrics import plot_confusion_matrix, accuracy_score
plot_confusion_matrix(model, x_test, y_test,cmap= "Blues")
print('Accuracy score: ',format(accuracy_score(y_test,y_pred_bagging)*100,'.2f'))
```

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm. A confusion matrix shows 4 values:

1. **True positive**
2. **True negative**
3. **False positive**
4. **False negative**

AUC ROC Curve:

```
from sklearn.metrics import roc_curve, auc


knn_fpr, knn_tpr, threshold = roc_curve(y_te, y_pred)
auc_knn = auc(knn_fpr, knn_tpr)*100

svm_fpr1, svm_tpr1, threshold = roc_curve(y_test, y_pred_svm)
auc_svm1 = auc(svm_fpr1, svm_tpr1)*100

plt.figure(figsize=(5,5), dpi=90)
plt.plot(knn_fpr, knn_tpr, linestyle='-.', label='KNN (auc = %0.2f)' % auc_knn)
plt.plot(svm_fpr1, svm_tpr1, linestyle='-', label='SVM (auc = %0.2f)' % auc_svm1)

plt.title('KNN V/S SVM with PCA')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

plt.legend()
plt.show()
```

AUC (Area Under The Curve) ROC (Receiver Operating Characteristics)

It is one of the most important evaluation metrics for checking any classification model's performance. It is also written as AUROC (Area Under the Receiver Operating Characteristics)

AUC - ROC curve is a performance measurement for classification problems at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much a model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s.

Algorithms used:

NLP:

Short for natural language processing, NLP focused on the enabling the computers to understand and interpret human language.

For any model we need to do some basic steps like importing basic libraries, importing dataset and extracting the x and y.

Refining data is very important in the NLP model. We used NLTK(Natural language processing toolkit) ,which provides a set of diverse natural languages algorithms.

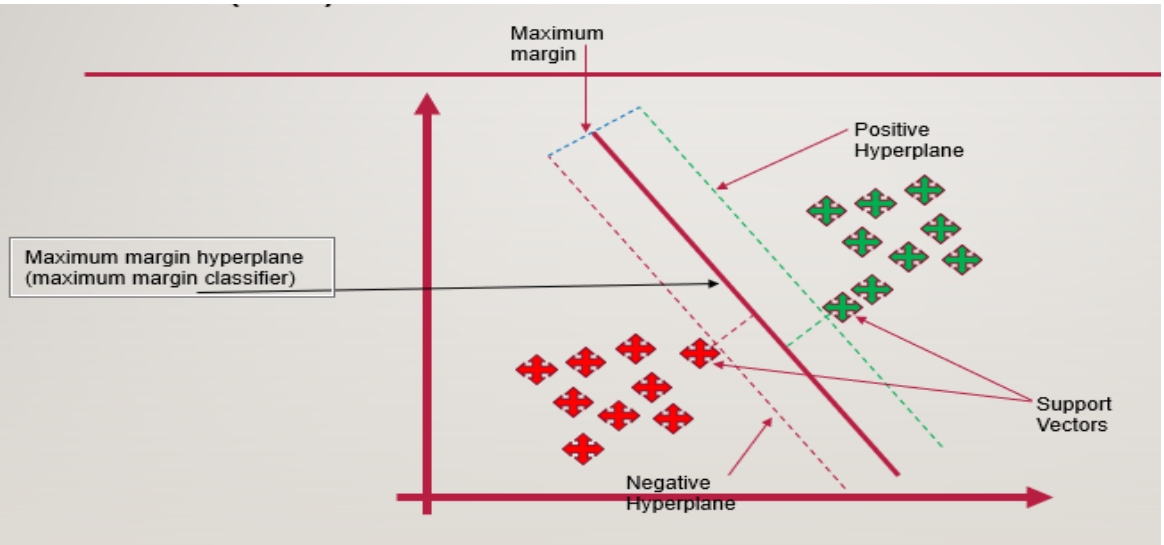
NLTK helps the computer to analysis, preprocess and understand the written text. In our model refining data involves the following steps:

1. **Replacing the punctuations with space**
2. **Change all text to lowercase**
3. **Removing stop words**
4. **Reducing words to their root word**
5. **Collecting the final refined reviews by using corpus**

Stopwords to remove are: ‘is’, ‘a’, ‘the’, ‘an’, ‘in’ and so on. Stop words are a set of commonly used words in any language.
Creating Bag of words: This model is the simplest way of extracting features from the text. This converts text into the matrix of occurrence of words within the document. The corpus will be split into training and test datasets. The training data set will be used to fit the model and the predictions on the test dataset.

SVM: (Support Vector Machine):

This is a supervised machine learning algorithm that can be used for both classification and regression challenges. Classification is predicting a label or group and Regression is predicting a continuous value. SVM plots all the numeric vectors in space and defines decision boundaries by hyper planes. This hyper plane separates the vectors in two categories such that the distance between each category from the hyper plane is maximum. The tuning parameter kernel – linear is used for this model. After training the model, a confusion matrix is generated which shows the number of positive and negative reviews, that are correctly predicted and which are wrongly predicted.



It is observed that SVM classifier outperforms every other classifier in predicting the reviews, as SVM provides us with an accuracy of 93.5. SVM performs exceptionally well with the prediction of reviews for the NLP model.

KNN: (K-Nearest Neighbor)

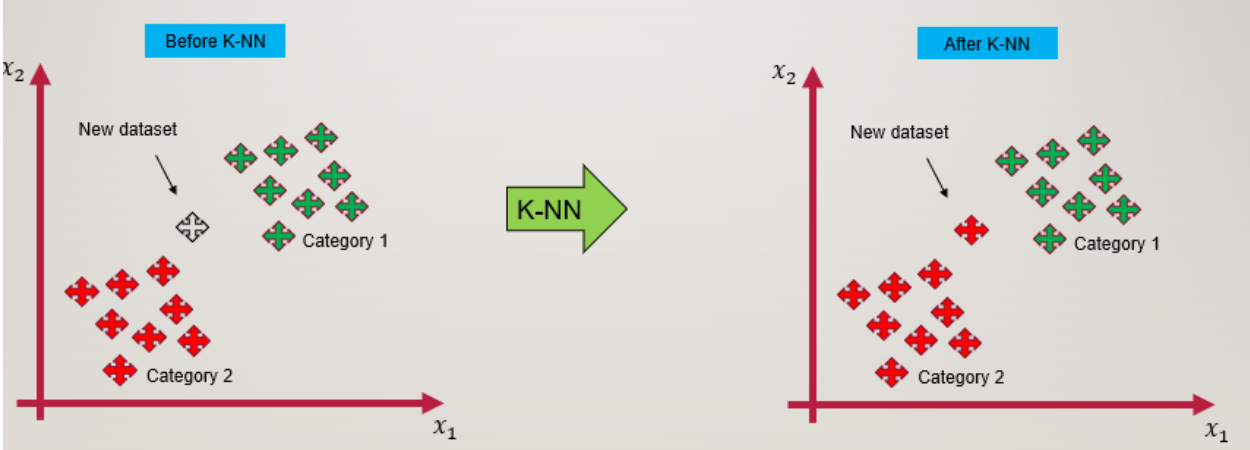
The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

Advantages:

1. **The algorithm is simple and easy to implement.**
2. **There's no need to build a model, tune several parameters, or make additional assumptions.**
3. **The algorithm is versatile. It can be used for classification, regression, and search.**

Disadvantages:

1. **The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase.**



K Nearest Neighbour is a classification algorithm being trained on Restaurants review data set in the current model. Best results are observed with metrics configured as 'minkowski', along with a p value of 6. The best nearest neighbour value is detected by plotting a line graph which displays the range of k values on X-axis and accuracy on Y-axis. The current best k value is expected to be 3, providing an accuracy of 92%.

Principal Component Analysis(PCA):

Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space.

It is a Unsupervised Machine Learning

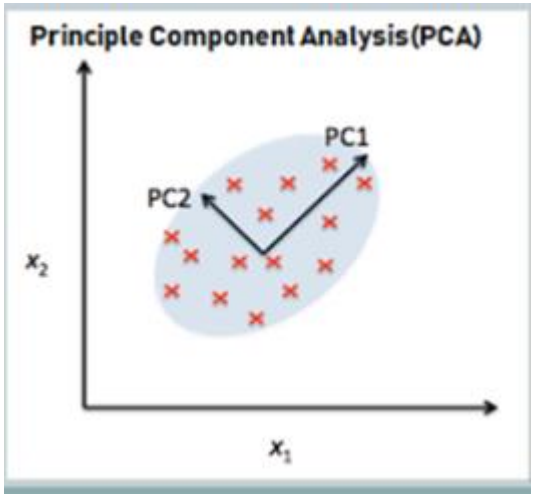
A transformation of your data and attempts to find out what features explain the most variance in your data.

Advantages of Principal Component Analysis

1. **Removes Correlated Features**
2. **Improves Algorithm Performance:**
3. **Reduces Overfitting**
4. **Improves Visualization**

Disadvantages

1. **Independent variables become less interpretable**
2. **Data standardization is must before PCA**



Hence forth , when pca is applied to our model of SVM we get the highest accuracy of 85.2 whereas , SVM gives us an accuracy of 93.5. This is because of the NLP technique applied on the dataset as PCA fails to differentiate

between the values of the features as our dependent and independent variables are an array of [0,1].

e.Conclusion:

Thus, in this project an efficient and user-friendly method is proposed which will provide automated systems in the restaurant and solve problems faced by the restaurants using technologies like Machine Learning and data science classification . Interactive User Interfaces for the customers and restaurant staff will be provided and customers can order food directly through the module without interacting with the waiter. Using Machine Learning Models prediction of the food preferred by the customers and also information necessary for the restaurant to grow in business through customer reviews and other data. The system saves a lot of time of the customer as well as the restaurant staff and helps the restaurant in many ways.

Based on the 3 previous mentioned models built by us , SVM provides us with the highest accuracy of 93.5% that itself indicates that our SVM model works best with natural language processing technique where the reviews of customers are processed in a detailed manner with the help of regularization , lemmatization , tokenization and creating a bag of words. We train our model with a test size of 20% of our entire dataset and using a confusion matrix find true positives and false positives.

f. Complete code :

1. Support Vector Machine (SVM)

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

## Importing dataset
df = pd.read_csv('Restaurant_Reviews.tsv', delimiter='\t', quoting=3)
print(df)

## Extracting x and y
x = df.iloc[:,:-1].values
y = df.iloc[:,-1].values

## Refining Data
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
corpus = [] # list will contain all the refined reviews

from stop_words import get_stop_words
from stop_words import safe_get_stop_words
for i in range(0,1000):
    review = df['Review'][i] # Collecting the reviews one by one
    review = re.sub('[^a-zA-z]', ' ', review) # replacing the punctuations with
space
    review = review.lower() # converting all the characters into lower case
    review = review.split() # splitting word of the statement i.e converting a
statement into list of words
    ps = PorterStemmer() # creating the object of the porter stemmer class
    # collecting the english language stop words
```

```
    all_stopwords = stopwords.words('english')
     all_stopwords +=safe_get_stop_words('unsupported language')

    # removing 'not' from the stopwords
    all_stopwords.remove('not')
     review =[ps.stem(word) for word in review if not word in
set(all_stopwords)]
    # converting the list of words back to statement
    # for this we will use the join function
    review = ' '.join(review)
    corpus.append(review) # collecting the refined reviews
print(corpus)

## Creating a bag of words
from sklearn.feature_extraction.text import CountVectorizer
import sklearn
cv = CountVectorizer(max_features=2500)
# using the max_feature parameter of countVectorizer to limit the number of
columns in x
x = cv.fit_transform(corpus).toarray()
y = df.iloc[:,-1].values
len(x[0])

## Splitting Training and Testing data
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size = 0.75,
random_state = 79)


## Creating and Training our Model
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier
classifier= SVC(kernel='linear',random_state=79, C=100, gamma=0.1)
model = BaggingClassifier( base_estimator=classifier,
```

```
                    bootstrap=True,# set to False to use Pasting instead of
Bagging
                    n_estimators =100, # number of SVC models to create
                    max_samples=220,
                # each model is trained from randomly sampled 100 instances
                    random_state=79 #n_estimators=100 max_samples=205
                   )
model.fit(x, y)

## Predicting the output
y_pr  = model.predict(x_test)
from sklearn import metrics
print("Accuracy:",format(metrics.accuracy_score(y_test, y_pr)*100,'.2f'))
print("Precision:",format(metrics.precision_score(y_test, y_pr)*100,'.2f'))
print("Recall:",format(metrics.recall_score(y_test, y_pr)*100,'.2f'))

## Plotting Confusion matrix and Accuracy
# plotting the confusion matrix and he accuracy score
from sklearn import metrics
from sklearn.metrics import plot_confusion_matrix,
accuracy_score,confusion_matrix
plot_confusion_matrix(model, x_test, y_test)
print(confusion_matrix(y_test,y_pr))
print("Accuracy score: ",format(metrics.accuracy_score(y_test
,y_pr)*100,'.2f'))

## Visualising the output
%matplotlib inline
print('Percentage for default\n')
print(round(df.Liked.value_counts(normalize=True)*100,2))
round(df.Liked.value_counts(normalize=True)*100,2).plot(kind='bar')
plt.title('Percentage Distributions by review type')
plt.show()

from sklearn.neighbors import KNeighborsClassifier
```

```
x_tr1,x_te1,y_tr1,y_te1 = train_test_split(x,y,train_size = 0.80, random_state =
124)
classifier =
KNeighborsClassifier(n_neighbors=3,metric='minkowski',leaf_size=1,p=1)
classifier.fit(x,y)
y_pred = classifier.predict(x_te1)
from sklearn.metrics import roc_curve, auc
knn_fpr, knn_tpr, threshold = roc_curve(y_te1, y_pred)
auc_knn = auc(knn_fpr, knn_tpr)*100
svm_fpr1, svm_tpr1, threshold = roc_curve(y_test, y_pr)
auc_svm1 = auc(svm_fpr1, svm_tpr1)*100
plt.figure(figsize=(5, 5), dpi=90)
plt.plot(knn_fpr, knn_tpr, linestyle='-', label='KNN (auc = %0.2f)' %
auc_knn)
plt.plot(svm_fpr1, svm_tpr1, linestyle='-.', label='SVM (auc = %0.2f)' %
auc_svm1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.title("SVM V/S KNN")
plt.show()
```

2. K -Nearest neighbour classifier (KNN)

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

# importing data set
df=pd.read_csv('Restaurant_Reviews.tsv', delimiter = "\t",quoting=3)
print(df)
# extracting x & y
x = df.iloc[:,:-1].values
y = df.iloc[:,-1].values

# refining data
#refining data
import re
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
corpus = [] # list will contain all the refined reviews

for i in range(0,1000):
    review = df['Review'][i] # Collecting the reviews one by one
    review = re.sub('[^a-zA-z]', ' ', review) # replacing the punctuations with
space
    review = review.lower() # converting all the characters into lower case
    review = review.split() # splitting word of the statement i.e converting a
statement into list of words
    ps = PorterStemmer() # creating the object of the porter stemmer class
    all_stopwords = stopwords.words('english') # collecting the english
language stop words
    # removing 'not' from the stopwords
    all_stopwords.remove('not')
```

```
    review =[ps.stem(word) for word in review if not word in
set(all_stopwords)]
    # converting the list of words back to statement
    # for this we will use the join function
    review = ' '.join(review)
    corpus.append(review) # collecting the refined reviews
print(corpus)

#creating bag of words
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=2500)
# using the max_feature parameter of countVectorizer to limit the number of
columns in x
x = cv.fit_transform(corpus).toarray()
y = df.iloc[:,-1].values
len(x[0])

# Feature Scaling
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
sc = StandardScaler()
x = sc.fit_transform(x)
scaler = RobustScaler()
x = scaler.fit_transform(x)

# splitting x & y into training set & test set
#splitting x & y into training set & test set
from sklearn.model_selection import train_test_split
x_tr,x_te,y_tr,y_te = train_test_split(x,y,train_size = 0.8, random_state =2)
#rand=179,124 trainsize=0.761,0.8
#creating and training knn model
from sklearn.neighbors import KNeighborsClassifier
classifier =
KNeighborsClassifier(n_neighbors=3,metric='minkowski',leaf_size=1, p=6)
classifier.fit(x,y)
```

```
# predicting output
y_pred = classifier.predict(x_te)
from sklearn import metrics
print("Accuracy:",format(metrics.accuracy_score(y_te, y_pred)*100,'.2f'))
print("Precision:",format(metrics.precision_score(y_te, y_pred)*100,'.2f'))
print("Recall:",format(metrics.recall_score(y_te, y_pred)*100,'.2f'))

## plotting confusion matrix & accuracy score
from sklearn.metrics import plot_confusion_matrix, accuracy_score,
confusion_matrix
from sklearn import metrics

plot_confusion_matrix(estimator=classifier,X=x_te,y_true=y_te)
print(confusion_matrix ( y_te , y_pred ))
print("Accuracy:",format(metrics.accuracy_score(y_te, y_pred)*100,'.2f'))

%matplotlib inline
print('Percentage for default\n')
print(round (df.Liked.value_counts(normalize=True)*100,2))
round(df. Liked.value_counts(normalize=True)*100,2).plot(kind='bar')
plt.title('Percentage Distributions by review type', fontsize=15)
plt.show()



from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,test_size=0.25,
random_state = 79) #rand=79 train_size=0.819 test=0.25

from sklearn.decomposition import PCA
pca = PCA(n_components= 5)
```

```
from sklearn.svm import SVC
classi= SVC(kernel='linear',random_state=0)
classi.fit(x_train,y_train)
#predicting the output for SVM with pca
y_pred_svm  = classi.predict(x_test)
from sklearn.metrics import roc_curve, auc

knn_fpr, knn_tpr, threshold = roc_curve(y_te, y_pred)
auc_knn = auc(knn_fpr, knn_tpr)*100

svm_fpr1, svm_tpr1, threshold = roc_curve(y_test, y_pred_svm)
auc_svm1 = auc(svm_fpr1, svm_tpr1)*100

plt.figure(figsize=(5,5), dpi=90)
plt.plot(knn_fpr, knn_tpr, linestyle='-.', label='KNN (auc = %0.2f)' %
auc_knn)
plt.plot(svm_fpr1, svm_tpr1, linestyle='-', label='SVM (auc = %0.2f)' %
auc_svm1)

plt.title('KNN V/S SVM with PCA')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

plt.legend()
plt.show()


# Detecting best k value
# try K=1 through K=25 and record testing accuracy
k_range = range(1, 7)

# We can create Python dictionary using [] or dict()
scores = []

# We use a loop through the range 1 to 26
```

```
# We append the scores in the dictionary
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x, y)
    y_pred = knn.predict(x_te)
    scores.append(metrics.accuracy_score(y_te, y_pred))

#print(scores)

# import Matplotlib (scientific plotting library)
import matplotlib.pyplot as plt

# allow plots to appear within the notebook
%matplotlib inline

# plot the relationship between K and testing accuracy
# plt.plot(x_axis, y_axis)
plt.plot(k_range, scores)
plt.title('Highest K value', fontsize =15)
plt.xlabel('Value of K for KNN', fontsize = 10)
plt.ylabel('Testing Accuracy', fontsize =10)
```

3. SVM with PCA

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

## Importing dataset
df = pd.read_csv('Restaurant_Reviews.tsv', delimiter='\t', quoting=3)
print(df)

## Extracting x and y
x = df.iloc[:,:-1].values
y = df.iloc[:,-1].values

## Refining Data
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
corpus = []
#stopset = set(stopwords.words('english')) - set(('over', 'under', 'below',
'more', 'most', 'no', 'not', 'only', 'such', 'few', 'so', 'too', 'very', 'just', 'any',
'once'))# list will contain all the refined reviews
from stop_words import get_stop_words
from stop_words import safe_get_stop_words
for i in range(0,1000):
    review = df['Review'][i] # Collecting the reviews one by one
    review = re.sub('[^a-zA-z]', ' ', review) # replacing the punctuations with
space
    review = review.lower() # converting all the characters into lower case
    review = review.split()
     # splitting word of the statement i.e converting a statement into list of
words
    ps = PorterStemmer() # creating the object of the porter stemmer class
```

```
    # collecting the english language stop words
    all_stopwords = stopwords.words('english')
    all_stopwords +=safe_get_stop_words('unsupported language')
    # removing 'not' from the stopwords
    all_stopwords.remove('not')

    review =[ps.stem(word) for word in review if not word in
set(all_stopwords)]
    # converting the list of words back to statement
    #review = [ps.stem(word) for word in review if not word in stopset]
    # for this we will use the join function
    review = ' '.join(review)
    corpus.append(review) # collecting the refined reviews
print(corpus)
## Creating a bag of words
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=2500) #min_df = 2, max_df = 0.8
# using the max_feature parameter of countVectorizer to limit the number of
columns in x
x = cv.fit_transform(corpus).toarray()
y = df.iloc[:,-1].values
len(x[0])

## Splitting Training and Testing data
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,test_size=0.25,
random_state = 79)

## Applying PCA
from sklearn.decomposition import PCA
pca = PCA(n_components= 2)
x = pca.fit_transform(x)
print("variance ratio: ", pca.explained_variance_ratio_)
```

```
# kernel PCA
from sklearn.decomposition import KernelPCA

kpca = KernelPCA(n_components = 3, kernel = 'linear',tol=0.1,alpha=2.0)
#n_components=3 kernel='linear'
x = kpca.fit_transform(x)

## Creating and Training our Model
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier
classifier= SVC(kernel='linear' , C=1000, gamma=1) #c=1000 g=1 rand=31
accuracy=67.95
model = BaggingClassifier( base_estimator=classifier,
                          bootstrap=True,# set to False to use Pasting instead of
Bagging
                         n_estimators =100, # number of SVC models to create
                         max_samples=220,
                    # each model is trained from randomly sampled 100 instances
                         random_state=49 #n_estimators=100 max_samples=205
                        )
model.fit(x_train, y_train)


## Predicting the output
y_pred_bagging = model.predict(x_test) # predicting
from sklearn import metrics
print("Accuracy:",format(metrics.accuracy_score(y_test,
y_pred_bagging)*100,'.2f'))
print("Precision:",format(metrics.precision_score(y_test,
y_pred_bagging)*100,'.2f'))
print("Recall:",format(metrics.recall_score(y_test,
y_pred_bagging)*100,'.2f'))

## Plotting Confusion matrix and Accuracy
# plotting the confusion matrix and he accuracy score
```

```
from sklearn.metrics import plot_confusion_matrix, accuracy_score
plot_confusion_matrix(model, x_test, y_test,cmap= "Blues")
print('Accuracy score:
',format(accuracy_score(y_test,y_pred_bagging)*100,'.2f'))

## Visualising the output
%matplotlib inline
print('Percentage for default\n')
print(round(df.Liked.value_counts(normalize=True)*100,2))
round(df.Liked.value_counts(normalize=True)*100,2).plot(kind='bar')
plt.title('Percentage Distributions by review type')
plt.show()
from sklearn.neighbors import KNeighborsClassifier
x_tr1,x_te1,y_tr1,y_te1 = train_test_split(x,y,train_size = 0.80, random_state =
124) #tr=0.71,ranst=13
classifier1 =
KNeighborsClassifier(n_neighbors=3,metric='minkowski',leaf_size=1,p=1)
#p=1
classifier1.fit(x,y)
y_pred = classifier1.predict(x_te1)
from sklearn.metrics import roc_curve, auc

knn_fpr, knn_tpr, threshold = roc_curve(y_te1, y_pred)
auc_knn = auc(knn_fpr, knn_tpr)*100

svm_fpr1, svm_tpr1, threshold = roc_curve(y_test, y_pred_bagging)
auc_svm1 = auc(svm_fpr1, svm_tpr1)*100

plt.figure(figsize=(5, 5), dpi=90)
plt.plot(knn_fpr, knn_tpr, linestyle='-', label='KNN (auc = %0.2f)' %
auc_knn)
plt.plot(svm_fpr1, svm_tpr1, linestyle='-.', label='SVM (auc = %0.2f)' %
auc_svm1)
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

plt.legend()
plt.title("SVM with PCA V/S KNN")
plt.show()
```