NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

Pattern Recognition and Recommender Systems (INITE69)

Practical File

Name: Dhruv Kumar Singh

Roll Number: 2021UIN3324

Branch: ITNS

Semester: 7

Submitted to : Priti Bansal

INDEX

Practical 1
Prepare webpage in Flask for some application

App.py

```
from flask import Flask, request, redirect, render_template

app = Flask(__name__)

@app.route("/")
def index():
    return render_template('index.html')

@app.route('/showGreeting', methods = ['POST'])
def result():
    userName = request.form['username']
    return render_template('showGreeting.html', name = userName)

if __name__ == "__main__":
    app.run(debug=True,host='0.0.0.0')
```

Index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Flask application</title>
    <link
      rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.100.2/css/materialize.min
.css"/>
  </head>
  <body>
    <div class="navbar-fixed">
      <nav>
        <div class="nav-wrapper">
          <span class="brand-logo">
            This is a greetings webapp, Enter your name to get greeting
          </span>
        </div>
      </nav>
    </div>
    <div class="row">
      <form class="col s12" action="/showGreeting" method="POST">
        <div class="row">
          <div class="input-field col s12">
            <input name="username" id="username" placeholder="Enter your name"
              type="text" class="validate"/>
          </div>
        </div>
```

```
        <button class="btn" type="submit" name="action">
          Submit
        </button>
      </form>
    </div>
  </body>
  <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
integrity="sha384KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
crossorigin="anonymous" ></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.100.2/js/
materialize.min.js"></script>
</html>
```

showGreeting.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Flask application</title>
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.100.2/css/materialize.min.css"
/>
  </head>
  <body>
    <div class="navbar-fixed">
      <nav>
        <div class="nav-wrapper">
          <span class="brand-logo">
            This is a greetings webapp, Enter your name to get greeting
          </span>
        </div>
      </nav>
    </div>
    <hr /> <h1>Hello {{name}}</h1> <hr />
  </body>
  <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
crossorigin="anonymous"></script>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.100.2/js/materialize.min.js"></s
cript>
</html>
```

Practical 2

Write a program which will implement Bayesian classification having single features, Multiple values and multiple classes for some application

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.feature_extraction.text import CountVectorizer
```

```
# Sample dataset with messages and their labels (spam or ham)
data = [
    ("Free money offer just for you", "spam"),
    ("Hello, let's catch up soon", "ham"),
    ("Win big prizes, claim now", "spam"),
    ("Meeting at 10am tomorrow", "ham"),
    ("Congratulations, you've won a free gift", "spam"),
    ("Are you free this weekend?", "ham"),
    ("Urgent! Claim your reward", "spam"),
    ("Call me when you are free", "ham"),
    ("Get cash prizes instantly", "spam"),
    ("How are you doing today?", "ham"),
    ("Limited offer! Don't miss out", "spam"),
    ("Can we reschedule our meeting?", "ham"),
    ("Congratulations, you have won a prize!", "spam"),
    ("Please call me when you have time", "ham"),
    ("Earn extra cash working from home", "spam"),
    ("Just checking in to say hi", "ham"),
    ("Exclusive deal for loyal customers", "spam"),
    ("Will you join us for dinner?", "ham"),
    ("Hurry! Offer expires soon", "spam"),
    ("Let's go for a walk tomorrow", "ham"),
    ("Important! Your account is at risk", "spam"),
    ("Hope you are doing well", "ham"),
    ("Click to claim your free voucher", "spam"),
    ("Let's plan for the weekend", "ham"),
    ("Special promotion just for you", "spam"),
    ("See you at the usual spot?", "ham"),
    ("Don't miss out on this chance", "spam"),
    ("Want to grab a coffee later?", "ham"),
    ("Redeem your reward points now", "spam"),
    ("Looking forward to catching up", "ham"),
    ("Act now to secure your winnings", "spam"),
    ("Are you free for lunch?", "ham")
]
```

```
# Split messages and labels
messages, labels = zip(*data)

# Convert text data into feature vectors using bag of words
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(messages)
y = list(labels)
```

```
# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Train Naive Bayes classifier
classifier = MultinomialNB()
classifier.fit(X_train, y_train)
```

```python
# Predict on test set and evaluate
y_pred = classifier.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred, zero_division=1))
```

```
Accuracy: 0.7
Classification Report:
              precision    recall  f1-score   support

         ham       0.62      1.00      0.77         5
        spam       1.00      0.40      0.57         5

    accuracy                           0.70        10
   macro avg       0.81      0.70      0.67        10
weighted avg       0.81      0.70      0.67        10
```

```python
# Predict a new message
new_message = ["Free entry in a contest, win big now"]
new_vector = vectorizer.transform(new_message)
prediction = classifier.predict(new_vector)
print("Predicted class for new message:", prediction[0])
```

```
Predicted class for new message: spam
```

Practical 3

Write a program which will implement Naïve Bayesian classification in NLP domain, also mention the steps for preprocessing

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.feature_extraction.text import CountVectorizer
```

```
# Sample dataset with messages and their labels (spam or ham)
data = [
    ("Free money offer just for you", "spam"),
    ("Hello, let's catch up soon", "ham"),
    ("Win big prizes, claim now", "spam"),
    ("Meeting at 10am tomorrow", "ham"),
    ("Congratulations, you've won a free gift", "spam"),
    ("Are you free this weekend?", "ham"),
    ("Urgent! Claim your reward", "spam"),
    ("Call me when you are free", "ham"),
    ("Get cash prizes instantly", "spam"),
    ("How are you doing today?", "ham"),
    ("Limited offer! Don't miss out", "spam"),
    ("Can we reschedule our meeting?", "ham"),
    ("Congratulations, you have won a prize!", "spam"),
    ("Please call me when you have time", "ham"),
    ("Earn extra cash working from home", "spam"),
    ("Just checking in to say hi", "ham"),
    ("Exclusive deal for loyal customers", "spam"),
    ("Will you join us for dinner?", "ham"),
    ("Hurry! Offer expires soon", "spam"),
    ("Let's go for a walk tomorrow", "ham"),
    ("Important! Your account is at risk", "spam"),
    ("Hope you are doing well", "ham"),
    ("Click to claim your free voucher", "spam"),
    ("Let's plan for the weekend", "ham"),
    ("Special promotion just for you", "spam"),
```

```
# Split messages and labels
messages, labels = zip(*data)

# Convert text data into feature vectors using bag of words
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(messages)
y = list(labels)
```

```
# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Train Naive Bayes classifier
classifier = MultinomialNB()
classifier.fit(X_train, y_train)
```

```
# Predict on test set and evaluate
y_pred = classifier.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred, zero_division=1))
```

```
Accuracy: 0.7
Classification Report:
              precision    recall  f1-score   support

         ham       0.62      1.00      0.77         5
        spam       1.00      0.40      0.57         5

    accuracy                           0.70        10
   macro avg       0.81      0.70      0.67        10
weighted avg       0.81      0.70      0.67        10
```

```python
# Predict a new message
new_message = ["Free entry in a contest, win big now"]
new_vector = vectorizer.transform(new_message)
prediction = classifier.predict(new_vector)
print("Predicted class for new message:", prediction[0])
```

```
Predicted class for new message: spam
```

Preprocessing Steps

1. **Clean Text**:
    - Remove punctuation and special characters.
    - Convert all text to lowercase.
2. **Remove Stop Words**:
    - Eliminate common words like "the" and "is" that don't add much meaning. Use a standard stop word list.
3. **Tokenize & Stem**:
    - Split the text into individual words (tokens).
    - Reduce words to their root form (e.g., "running" → "run").
4. **Convert Text to Numerical Data**:
    - Use **CountVectorizer** or **TfidfVectorizer** to turn text data into numbers that the model can understand.
5. **Label Encoding**:
    - Convert text labels (like "spam" and "ham") into numerical values.
6. **Train-Test Split**:
    - Split the data into training and test sets (e.g., 70% training, 30% testing) for model evaluation.

Practical 4

Write a program to implement Bayesian Risk for some application

```
import numpy as np

prior_disease = 0.01
prior_no_disease = 1 - prior_disease

likelihood_pos_given_disease = 0.9
likelihood_neg_given_no_disease = 0.95

loss_disease_given_no_disease = 10
loss_no_disease_given_disease = 100

def posterior_probabilities(prior, likelihood_pos, likelihood_neg):
    prob_positive = (likelihood_pos * prior) + ((1 - likelihood_neg) * (1 - prior))
    prob_negative = ((1 - likelihood_pos) * prior) + (likelihood_neg * (1 - prior))

    posterior_disease_given_pos = (likelihood_pos * prior) / prob_positive
    posterior_no_disease_given_pos = ((1 - likelihood_neg) * (1 - prior)) / prob_positive

    posterior_disease_given_neg = ((1 - likelihood_pos) * prior) / prob_negative
    posterior_no_disease_given_neg = (likelihood_neg * (1 - prior)) / prob_negative

    return {
        'P(D|+ Test)': posterior_disease_given_pos,
        'P(~D|+ Test)': posterior_no_disease_given_pos,
        'P(D|- Test)': posterior_disease_given_neg,
        'P(~D|- Test)': posterior_no_disease_given_neg
    }

def calculate_bayesian_risk(posteriors):
    risk_predict_disease = (posteriors['P(D|+ Test)'] * 0) + (posteriors['P(~D|+ Test)'] *
loss_disease_given_no_disease)

    risk_predict_no_disease = (posteriors['P(D|+ Test)'] * loss_no_disease_given_disease)
+ (posteriors['P(~D|+ Test)'] * 0)

    return {
        'Risk(Predict Disease | + Test)': risk_predict_disease,
        'Risk(Predict No Disease | + Test)': risk_predict_no_disease
    }

posteriors = posterior_probabilities(prior_disease, likelihood_pos_given_disease,
likelihood_neg_given_no_disease)
risk = calculate_bayesian_risk(posteriors)

print("Posterior Probabilities for + Test:")
print(posteriors)
print("\nBayesian Risk for Decisions with + Test:")
print(risk)
```

Practical 5

Implement Principal Component Analysis for some application

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
```

```
from sklearn.datasets import load_breast_cancer as lbc
cancer = lbc()
```

```
cancer.keys()
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

```
print(cancer['DESCR'])
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class
```

```
df = pd.DataFrame(cancer['data'], columns=cancer['feature_names'])
```

```
df.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.1622
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.1238
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.1444
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.2098
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.1374

5 rows × 30 columns

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
scaler.fit(df)
```

```
StandardScaler()
```

```
scaled_data = scaler.transform(df)
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
```

```
pca.fit(scaled_data)
```

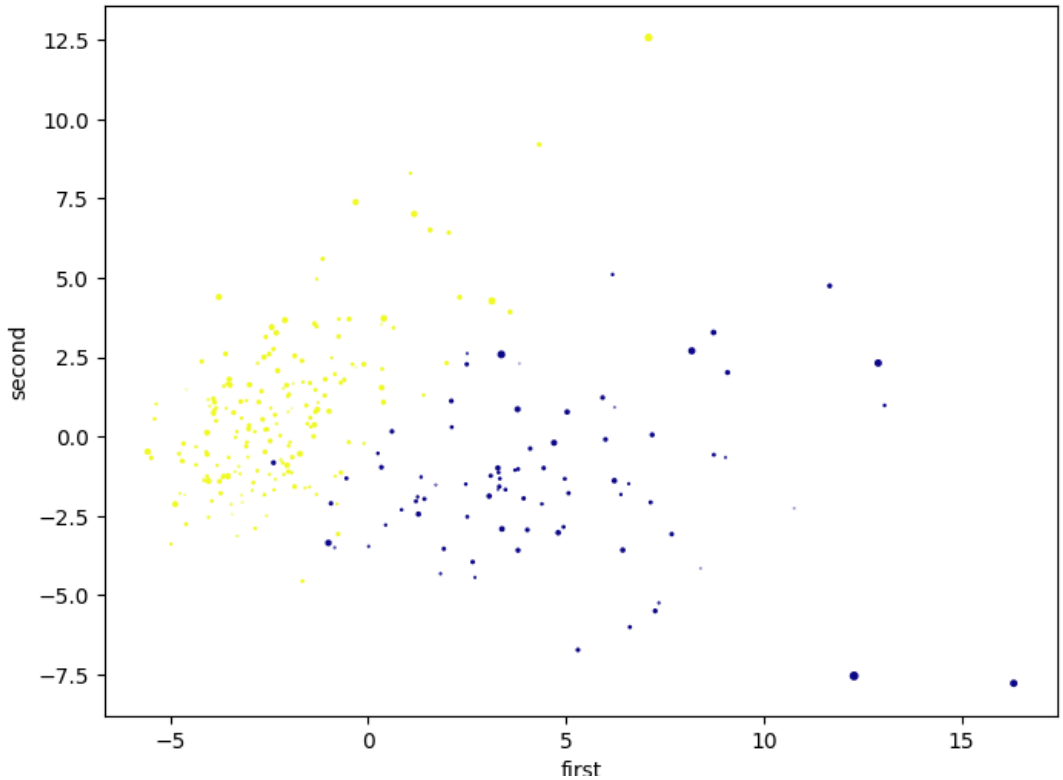
```
PCA(n_components=3)
```

```
xpca = pca.transform(scaled_data)
```

```
xpca.shape
```

```
(569, 3)
```

```
plt.figure(figsize=(8,6))
plt.scatter(xpca[:,0],xpca[:,1],c=cancer['target'],cmap='plasma')
plt.xlabel('first')
plt.ylabel('second')
```



Practical 6

Write a program to generate content based recommender system for movies

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Load the movie dataset
moviesdf = pd.read_csv('./ml-latest-small/movies.csv', usecols=['movieId', 'title', 'genres'])
```

```
# Replace '|' in genres with spaces so they are treated as separate words
moviesdf['genres'] = moviesdf['genres'].str.replace('|', ' ')

# Use TF-IDF to transform the genres column
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(moviesdf['genres'])
```

```
moviesdf.head()
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
1	2	Jumanji (1995)	Adventure Children Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama Romance
4	5	Father of the Bride Part II (1995)	Comedy

```
# Compute cosine similarity matrix
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
```

```
# Reset the movie indices to make it easier to find movies by index
indices = pd.Series(moviesdf.index, index=moviesdf['title']).drop_duplicates()

def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices[title]

    # Get the pairwise similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return moviesdf['title'].iloc[movie_indices]
```

```
# Example usage
print(get_recommendations('Toy Story (1995)'))
```

```
1706                                      Antz (1998)
2355                             Toy Story 2 (1999)
2809       Adventures of Rocky and Bullwinkle, The (2000)
3000                  Emperor's New Groove, The (2000)
3568                            Monsters, Inc. (2001)
6194                                 Wild, The (2006)
6486                           Shrek the Third (2007)
6948                 Tale of Despereaux, The (2008)
7760    Asterix and the Vikings (Astérix et les Viking...
8219                                      Turbo (2013)
Name: title, dtype: object
```