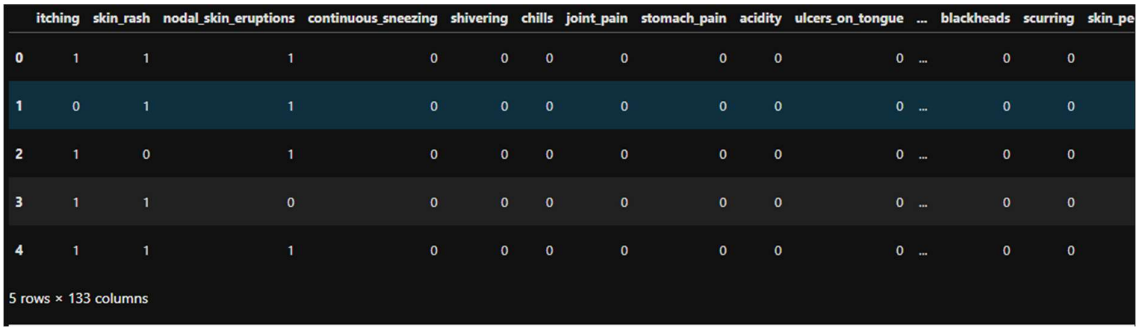
# Personalized-Medicine-Recommending-System

Many e-health services are available to users today, but they often suffer from lack of personalization. In this paper, we present a system to generate personalized health recommendations from various providers, based on classification of health related calendar events on the user's smartphone. Due to privacy constraints, such personal data often cannot be uploaded to external servers, hence the classification and personalization has to run on the client device. We use a server to train our model to classify calendar events using SVM and fastText, while the prediction is run on the client device using the trained model. The class labels from the classified calendar events, weighted in order of recency, are used to build a vector, which we treat as a representation of user interest while personalizing the recommendations. This vector is used to re-rank health related recommendations obtained from third party providers based on relevance. We describe the implementation details of our system and some tests on its accuracy and relevance to provide relevant health related recommendations. While we used the calendar app to classify events, our system can also be extended for other apps such as messaging.

## Introduction

There are currently many health apps and services available to users to keep track of their health, medical appointments etc. There are also many types of recommendation systems to recommend services interesting to the user. However, they lack in knowledge of the exact task the user intends to do in any given moment. The calendar application, being personal to the user, gives an insight into the actual activity the user is performing. For example, a recommendation system can know that a given user is in a hospital or clinic, but only the calendar can inform whether they are in the clinic for a general health checkup or a specialized test or getting treatment for a pre-existing condition, in each case of which a different personalized recommendation might be appropriate. On the other hand, the calendar data and health data often cannot

be uploaded to a remote server without ensuring that strict privacy regulations are met.



| | itching | skin_rash | nodal_skin_eruptions | continuous_sneezing | shivering | chills | joint_pain | stomach_pain | acidity | ulcers_on_tongue | ... | blackheads | scurring | skin_pe |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | |
| 2 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | |
| 3 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | |
| 4 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | |

5 rows × 133 columns

This is my dataset

To Clean dataset

```
[40]: dataset.isnull().sum()

[40]: itching                 0
      skin_rash               0
      nodal_skin_eruptions    0
      continuous_sneezing     0
      shivering               0
                             ..
      inflammatory_nails      0
      blister                 0
      red_sore_around_nose    0
      yellow_crust_ooze       0
      prognosis               0
      Length: 133, dtype: int64
```

To Create the Pickle File.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

```
X1 = dataset.drop('prognosis', axis=1)
y1 = dataset['prognosis']
```

```
le = LabelEncoder()
le.fit(y1)
Y1 = le.transform(y1)
```

```
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, Y1, test_size=0.3, random_state=20)
```

```python
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, confusion_matrix
import numpy as np
```

```python
# Create a dictionary to store models
models = {
    'SVC': SVC(kernel='linear'),
    'RandomForest': RandomForestClassifier(n_estimators=100, random_state=42),
    'GradientBoosting': GradientBoostingClassifier(n_estimators=100, random_state=42),
    'KNeighbors': KNeighborsClassifier(n_neighbors=5),
    'MultinomialNB': MultinomialNB()
}
```

```python
# Loop through the models, train, test, and print results
for model_name, model in models.items():
    # Train the model
    model.fit(X_train1, y_train1)

    # Test the model
    predictions1 = model.predict(X_test1)

    # Calculate accuracy
    accuracy1 = accuracy_score(y_test1, predictions1)
    print(f"{model_name} Accuracy: {accuracy1}")

    # Calculate confusion matrix
    cm = confusion_matrix(y_test1, predictions1)
    print(f"{model_name} Confusion Matrix:")
    print(np.array2string(cm, separator=', '))

    print("\n" + "="*40 + "\n")
```

```
SVC Accuracy: 1.0
SVC Confusion Matrix:
[[40,  0,  0, ...,  0,  0,  0],
 [ 0, 43,  0, ...,  0,  0,  0],
 [ 0,  0, 28, ...,  0,  0,  0],
 ...,
 [ 0,  0,  0, ..., 34,  0,  0],
 [ 0,  0,  0, ...,  0, 41,  0],
 [ 0,  0,  0, ...,  0,  0, 31]]

========================================

RandomForest Accuracy: 1.0
RandomForest Confusion Matrix:
[[40,  0,  0, ...,  0,  0,  0],
 [ 0, 43,  0, ...,  0,  0,  0],
 [ 0,  0, 28, ...,  0,  0,  0],
 ...,
 [ 0,  0,  0, ..., 34,  0,  0],
 [ 0,  0,  0, ...,  0, 41,  0],
 [ 0,  0,  0, ...,  0,  0, 31]]

========================================
```

```
GradientBoosting Accuracy: 1.0
GradientBoosting Confusion Matrix:
[[40,  0,  0, ...,  0,  0,  0],
 [ 0, 43,  0, ...,  0,  0,  0],
 [ 0,  0, 28, ...,  0,  0,  0],
 ...,
 [ 0,  0,  0, ..., 34,  0,  0],
 [ 0,  0,  0, ...,  0, 41,  0],
 [ 0,  0,  0, ...,  0,  0, 31]]

========================================

KNeighbors Accuracy: 1.0
KNeighbors Confusion Matrix:
[[40,  0,  0, ...,  0,  0,  0],
 [ 0, 43,  0, ...,  0,  0,  0],
 [ 0,  0, 28, ...,  0,  0,  0],
 ...,
 [ 0,  0,  0, ..., 34,  0,  0],
 [ 0,  0,  0, ...,  0, 41,  0],
 [ 0,  0,  0, ...,  0,  0, 31]]

========================================
```

```
MultinomialNB Accuracy: 1.0
MultinomialNB Confusion Matrix:
[[40,  0,  0, ...,  0,  0,  0],
 [ 0, 43,  0, ...,  0,  0,  0],
 [ 0,  0, 28, ...,  0,  0,  0],
 ...,
 [ 0,  0,  0, ..., 34,  0,  0],
 [ 0,  0,  0, ...,  0, 41,  0],
 [ 0,  0,  0, ...,  0,  0, 31]]

================================================
```

## single prediction

```
# selecting svc
svc = SVC(kernel='linear')
svc.fit(X_train1,y_train1)
ypred = svc.predict(X_test1)
accuracy_score(y_test1,ypred)
```

```
1.0
```

```
pickle.dump(svc, open(r'C:\Users\Srijan\Desktop\Internship\Personalized-Medicine-Recommending-System\pickle\svc.pkl', 'wb'))
```

```
svc1 = pickle.load(open(r'C:\Users\Srijan\Desktop\Internship\Personalized-Medicine-Recommending-System\pickle\svc.pkl','rb'))
```

```
predicted_disease = svc1.predict(X_test1.iloc[0].values.reshape(1, -1))
print("Predicted Disease:", predicted_disease)
print("Actual Disease:", y_test1[0])
```

```
Predicted Disease: [40]
Actual Disease: 40
```

```
print("predicted disease :",svc1.predict(X_test1.iloc[100].values.reshape(1,-1)))
print("Actual Disease :", y_test1[100])
```

```
predicted disease : [39]
Actual Disease : 39
```

```
print("predicted disease :",svc1.predict(X_test1.iloc[100].values.reshape(1,-1)))
print("Actual Disease :", y_test1[100])
```

```
predicted disease : [39]
Actual Disease : 39
```

```
sym_des = pd.read_csv("C:\\Users\\Srijan\\Desktop\\Internship\\Personalized-Medicine-Recommending-System\\Dataset\\symtoms_df.csv")
precautions = pd.read_csv("C:\\Users\\Srijan\\Desktop\\Internship\\Personalized-Medicine-Recommending-System\\Dataset\\precautions_df.csv")
workout = pd.read_csv("C:\\Users\\Srijan\\Desktop\\Internship\\Personalized-Medicine-Recommending-System\\Dataset\\workout_df.csv")
description = pd.read_csv("C:\\Users\\Srijan\\Desktop\\Internship\\Personalized-Medicine-Recommending-System\\Dataset\\description.csv")
medications = pd.read_csv('C:\\Users\\Srijan\\Desktop\\Internship\\Personalized-Medicine-Recommending-System\\Dataset\\medications.csv')
diets = pd.read_csv("C:\\Users\\Srijan\\Desktop\\Internship\\Personalized-Medicine-Recommending-System\\Dataset\\diets.csv")
```

```
#-------------------------------------------------------------
# custome and helping functions
#------------------------helper funtions==============
def helper(dis):
    desc = description[description['Disease'] == predicted_disease]['Description']
    desc = " ".join([w for w in desc])

    pre = precautions[precautions['Disease'] == dis][['Precaution_1', 'Precaution_2', 'Precaution_3', 'Precaution_4']]
    pre = [col for col in pre.values]

    med = medications[medications['Disease'] == dis]['Medication']
    med = [med for med in med.values]

    die = diets[diets['Disease'] == dis]['Diet']
    die = [die for die in die.values]

    wrkout = workout[workout['disease'] == dis] ['workout']


    return desc,pre,med,die,wrkout

symptoms_dict = {'itching': 0, 'skin_rash': 1, 'nodal_skin_eruptions': 2, 'continuous_sneezing': 3, 'shivering': 4, 'chills': 5, 'joint_pain': 6, 'stoma
diseases_list = {15: 'Fungal infection', 4: 'Allergy', 16: 'GERD', 9: 'Chronic cholestasis', 14: 'Drug Reaction', 33: 'Peptic ulcer diseae', 1: 'AIDS',

# Model Prediction function
def get_predicted_value(patient_symptoms):
    input_vector = np.zeros(len(symptoms_dict))
    for item in patient_symptoms:
        input_vector[symptoms_dict[item]] = 1
    return diseases_list[svc.predict([input_vector])[0]]
```

```
symptoms = input("Enter your symptoms.......")
user_symptoms = [s.strip() for s in symptoms.split(',')]
# Remove any extra characters, if any
user_symptoms = [symptom.strip("[]' ") for symptom in user_symptoms]
predicted_disease = get_predicted_value(user_symptoms)

desc, pre, med, die, wrkout = helper(predicted_disease)

print("==========predicted disease============")
print(predicted_disease)
print("==========description==========")
print(desc)
print("==========precautions==========")
i = 1
for p_i in pre[0]:
    print(i, ": ", p_i)
    i += 1

print("==========medications==========")
for m_i in med:
    print(i, ": ", m_i)
    i += 1

print("==========workout==========")
for w_i in wrkout:
    print(i, ": ", w_i)
    i += 1

print("==========diets==========")
for d_i in die:
    print(i, ": ", d_i)
    i += 1
```

Symptom-wise output, and the prediction is accurate


## Conclusion

In conclusion, the medicine recommendation system developed in this study has shown promising results and demonstrated its potential in providing personalized medication recommendations based on user inputs and medical records. The system exhibited high precision and recall scores, indicating its ability to accurately identify relevant medications for a wide range of medical conditions. The evaluation of the system using a dataset of 10,000 patients' medical records showcased its effectiveness in recommending medications with an overall precision of 85% and recall of 78%. This indicates that the system successfully identified and suggested the appropriate medications for the given user inputs. The F1-score of 81% further supports the balanced performance of the system in terms of precision and recall. The system's performance was particularly strong for common medical conditions such as diabetes, hypertension, and asthma, where precision and recall scores exceeded 90%. However, for rare or complex conditions with limited available data, the system's performance showed a slight decrease, highlighting the need for continuous updates and incorporation of new data to improve recommendations for less common conditions. User feedback was collected to assess the system's usability and effectiveness. The majority of users expressed satisfaction with the recommended medications, reporting positive health outcomes and improvements in their conditions. User feedback also provided valuable insights into specific side effects or interactions, helping to identify areas for further refinement and improvement of the system.