Disease Prediction using Machine Learning

By: Rhichard Koh

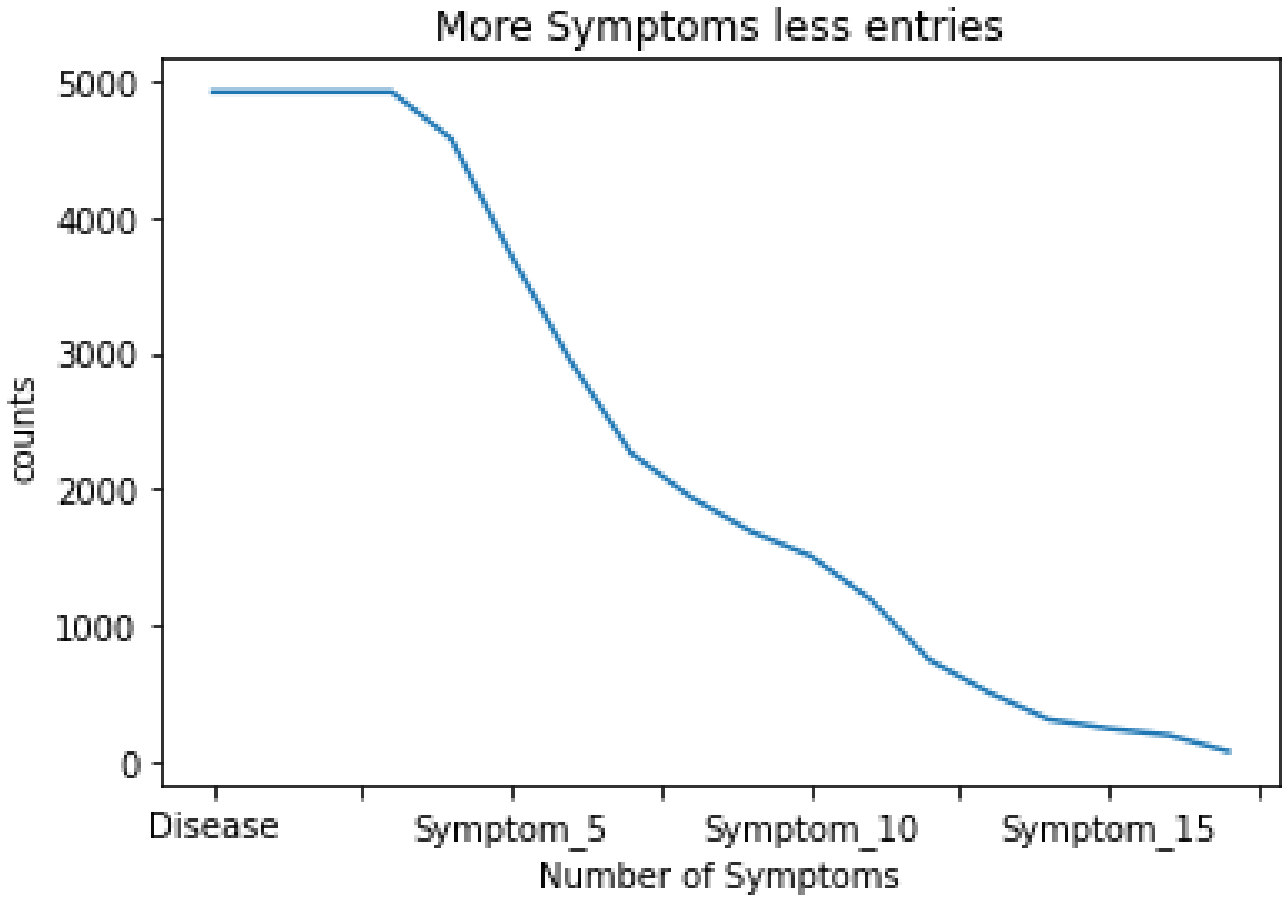
Introduction

- The problem that I will be tackling in my final project is disease prediction based on symptoms given.
- The motivation for this work is to help physicians diagnose people and it can be an aid for people that live in places where they do not have access to immediate healthcare.
- The benefits of running a machine learning model over this dataset rather than hard coding each disease and their symptoms are that it is more efficient to program, easier to maintain, and very easy to update with new diseases.

Other Solutions

- Hard coding each disease and their symptom
- This is very inefficient, takes a long time to do.
- Not very scalable compared to a machine learning solution.

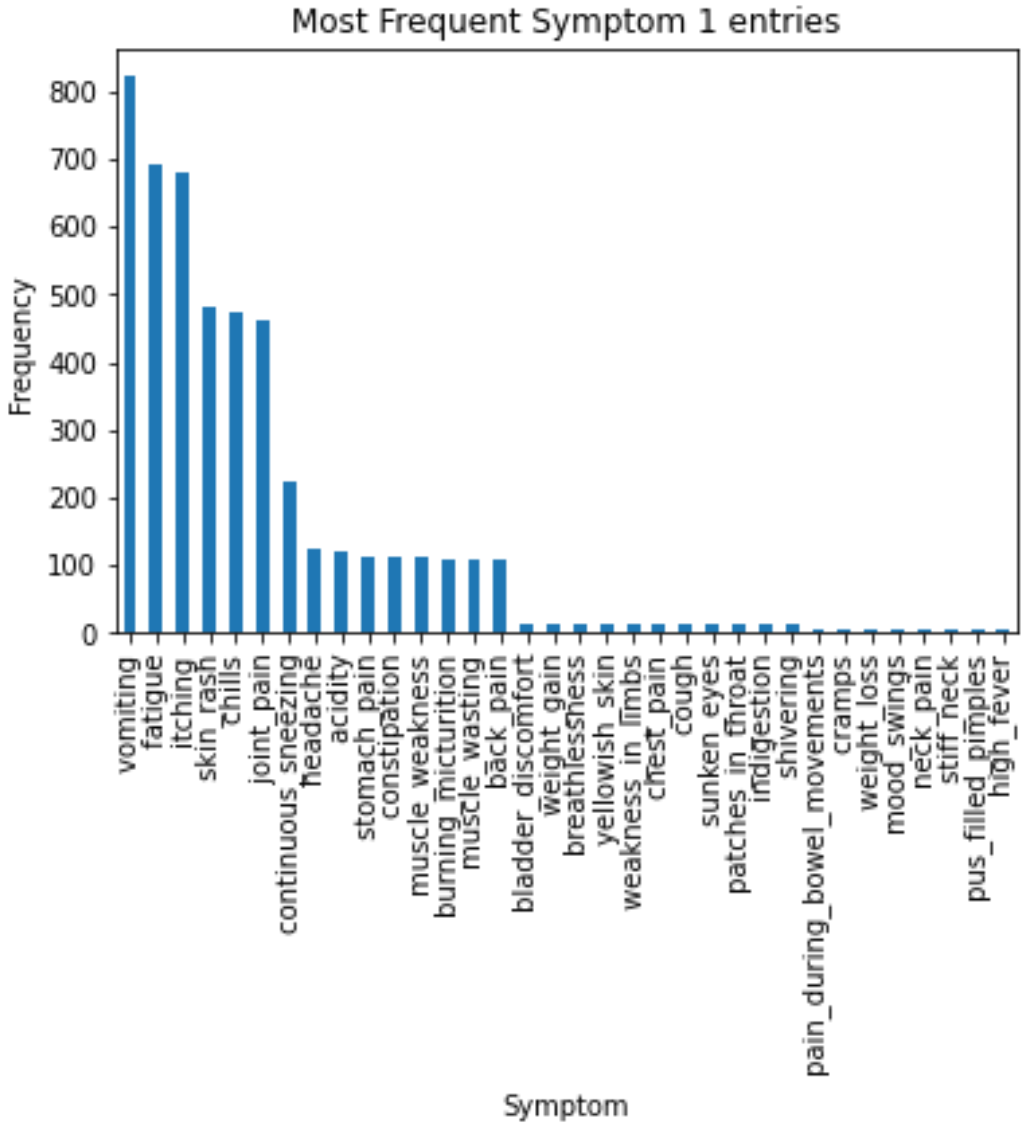
Exploratory Data Analysis



Raw data frame

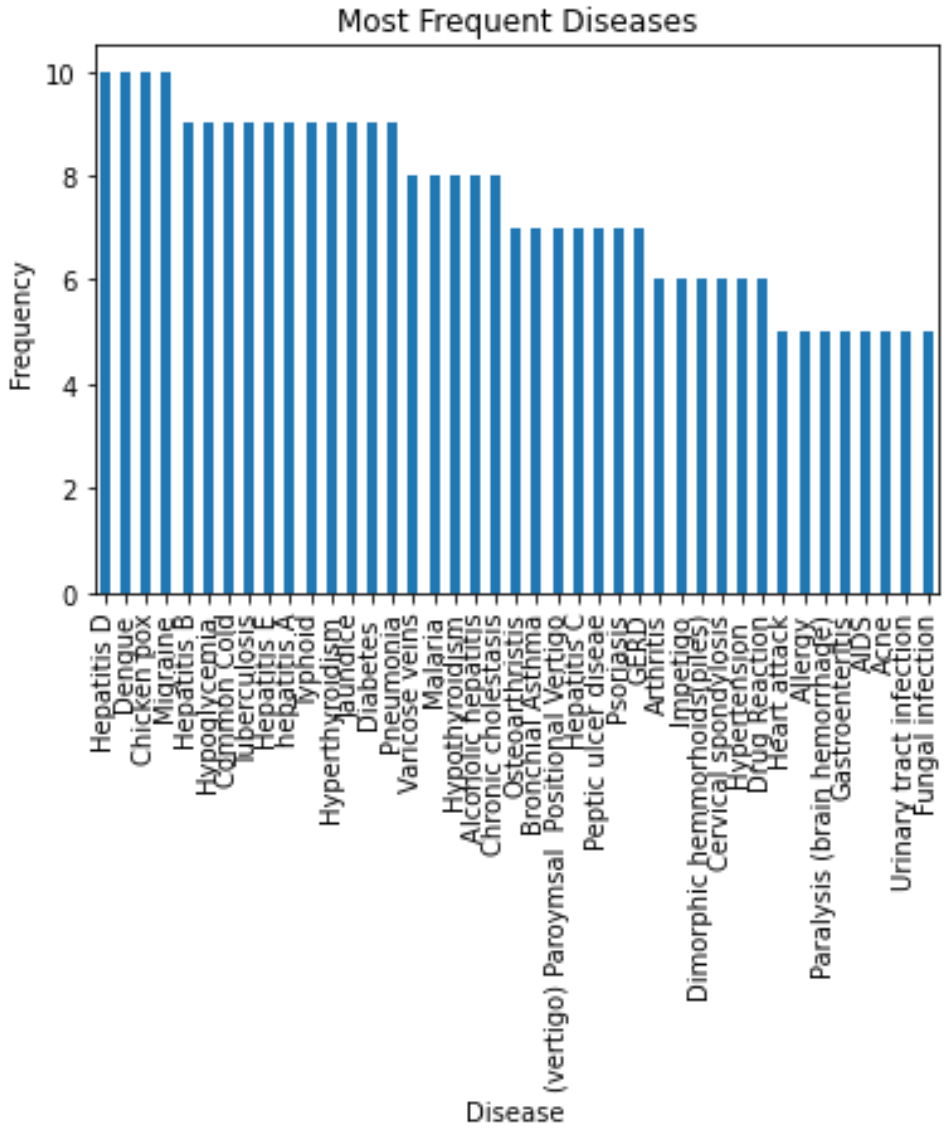
Exploratory Data Analysis

Most Frequent Symptom 1 entries from raw data frame

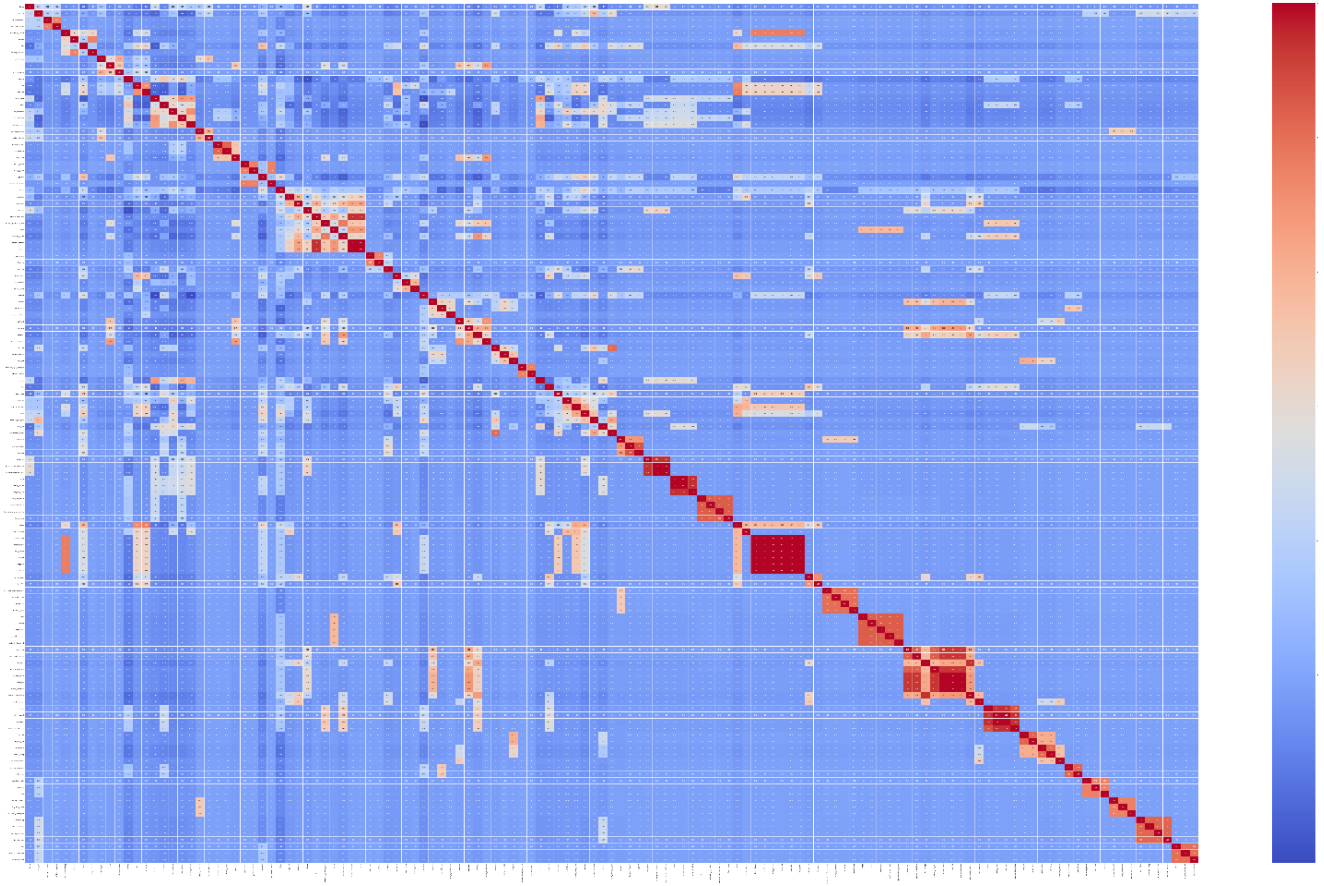


Exploratory Data Analysis

Most frequent disease from encoded data frame



Exploratory Data Analysis



Correlation Heat Map from encoded data frame

Experiment Design

Algorithms Used:

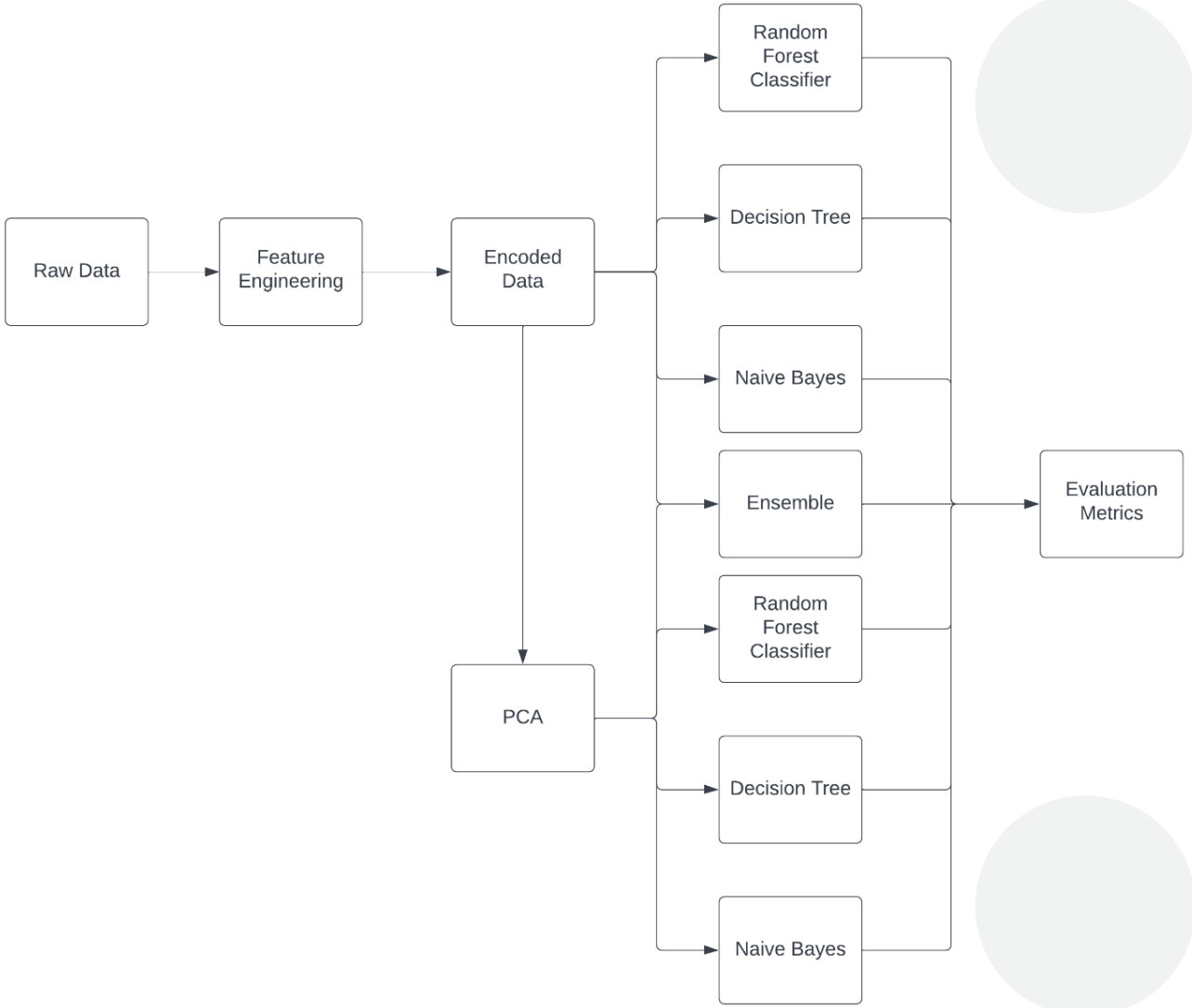
- Random Forest Classifier
- Decision Tree
- Naïve Bayes

Experiment Design

Evaluation Metrics Used:

- Accuracy Score
- Confusion Matrix
- F1 Score

Experiment Design



Experiment Design

Original Data

	Disease	Symptom_1	Symptom_2	Symptom_3	Symptom_4	Symptom_5	Symptom_6	Symptom_7	Symptom_8	Symptom_9	Symptom_10	Symptom_11	Symptom_12	Symptom_13	Symptom_14	Symptom_15
0	Fungal infection	itching	skin_rash	nodal_skin_eruptions	dischromic _patches	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Fungal infection	skin_rash	nodal_skin_eruptions	dischromic _patches	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	Fungal infection	itching	nodal_skin_eruptions	dischromic _patches	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	Fungal infection	itching	skin_rash	dischromic _patches	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	Fungal infection	itching	skin_rash	nodal_skin_eruptions	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4915	(vertigo) Paroymsal Positional Vertigo	vomiting	headache	nausea	spinning_movements	loss_of_balance	unsteadiness	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4916	Acne	skin_rash	pus_filled_pimples	blackheads	scurring	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4917	Urinary tract infection	burning_micturition	bladder_discomfort	foul_smell_of urine	continuous_feel_of_urine	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4918	Psoriasis	skin_rash	joint_pain	skin_peeling	silver_like_dusting	small_dents_in_nails	inflammatory_nails	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4919	Impetigo	skin_rash	high_fever	blister	red_sore_around_nose	yellow_crust_ooze	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

4920 rows × 19 columns

Experiment Design

Adding a symptom column and setting all the rows to 0

```
[ ]  df['Symptoms'] = 0
     df
```

Putting all the symptoms of each row in a list then replacing the 0s in the symptom column with the symptom list.

```
[ ]  for i in range(len(df)):
       vals = df.iloc[i].values
       vals = vals.tolist()
       if 0 in vals:
         df['Symptoms'][i] = vals[1:vals.index(0)]
       else:
         df['Symptoms'][i] = vals[1:]
     df
```

Unraveling the symptom column symptom lists.

```
[ ]  col_vals = df.iloc[:,1:-1].values.ravel()
     col_vals
```

Experiment Design

Getting all of the unique values of the symptom list, ignores if its nan.

```
[ ]  symptoms = pd.unique(col_vals)
     symptoms.tolist()
     symptoms = [i for i in symptoms if str(i) != 'nan']
     symptoms
```

Made a new dataframe with the symptom names as the column and same index as the previous dataframe.

```
[ ]  symptoms_df = pd.DataFrame(columns=symptoms, index=df.index)
     symptoms_df
```

Experiment Design

Feature Engineered Data

	itching	skin_rash	nodal_skin_eruptions	dischromic _patches	continuous_sneezing	shivering	chills	watering_from_eyes	stomach_pain	acidity	...	bladder_discomfort	foul_smell_of urine	continuous_feel_of_urine	skin_peeling	silver_like
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4915	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
4916	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
4917	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
4918	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
4919	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	

4920 rows × 131 columns

Experiment Design

Copied the symptom column and the symptom lists for each row from the previous dataframe.

```
[ ]  symptoms_df['Symptoms'] = df['Symptoms']
     symptoms_df
```

Populated the new dataframe with 1 if that symptom appears in the symptom column's symptom list or 0 if the symptom wasnt in the list for each row.

```
[ ]  for i in symptoms:
       symptoms_df[i] = symptoms_df.apply(lambda x:1 if i in x.Symptoms else 0, axis=1)
     symptoms_df
```

Experiment Design

Encoded Data

	itching	skin_rash	nodal_skin_eruptions	dischromic _patches	continuous_sneezing	shivering	chills	watering_from_eyes	stomach_pain	acidity	...	foul_smell_of urine	continuous_feel_of_urine	skin_peeling	silver_like_dusting	small_dents_
0	1	1	1	1	0	0	0	0	0	0	...	0	0	0	0	
1	0	1	1	1	0	0	0	0	0	0	...	0	0	0	0	
2	1	0	1	1	0	0	0	0	0	0	...	0	0	0	0	
3	1	1	0	1	0	0	0	0	0	0	...	0	0	0	0	
4	1	1	1	0	0	0	0	0	0	0	...	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4915	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
4916	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	
4917	0	0	0	0	0	0	0	0	0	0	...	1	1	0	0	
4918	0	1	0	0	0	0	0	0	0	0	...	0	0	1	1	
4919	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	

4920 rows × 132 columns

Experiment Design

I chose to encode the data manually because if I were to use the auto encoder, it would have added 400+ features instead of the 131 it gives when I did it manually.

Experiment Results

Random Forest Classifier

Training a Random Forest Classifier

```
[ ] from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import accuracy_score, confusion_matrix, f1_score

    rfc = RandomForestClassifier(criterion='entropy',n_estimators=500, max_leaf_nodes=16, n_jobs=-1, random_state=42)
    rfc.fit(X_train,y_train)
```

```
RandomForestClassifier(criterion='entropy', max_leaf_nodes=16, n_estimators=500,
                       n_jobs=-1, random_state=42)
```

Cross Validiation Score

```
[ ] cross_val_score(rfc,X_train,y_train,cv=3).mean()
```

```
0.9629629629629629
```

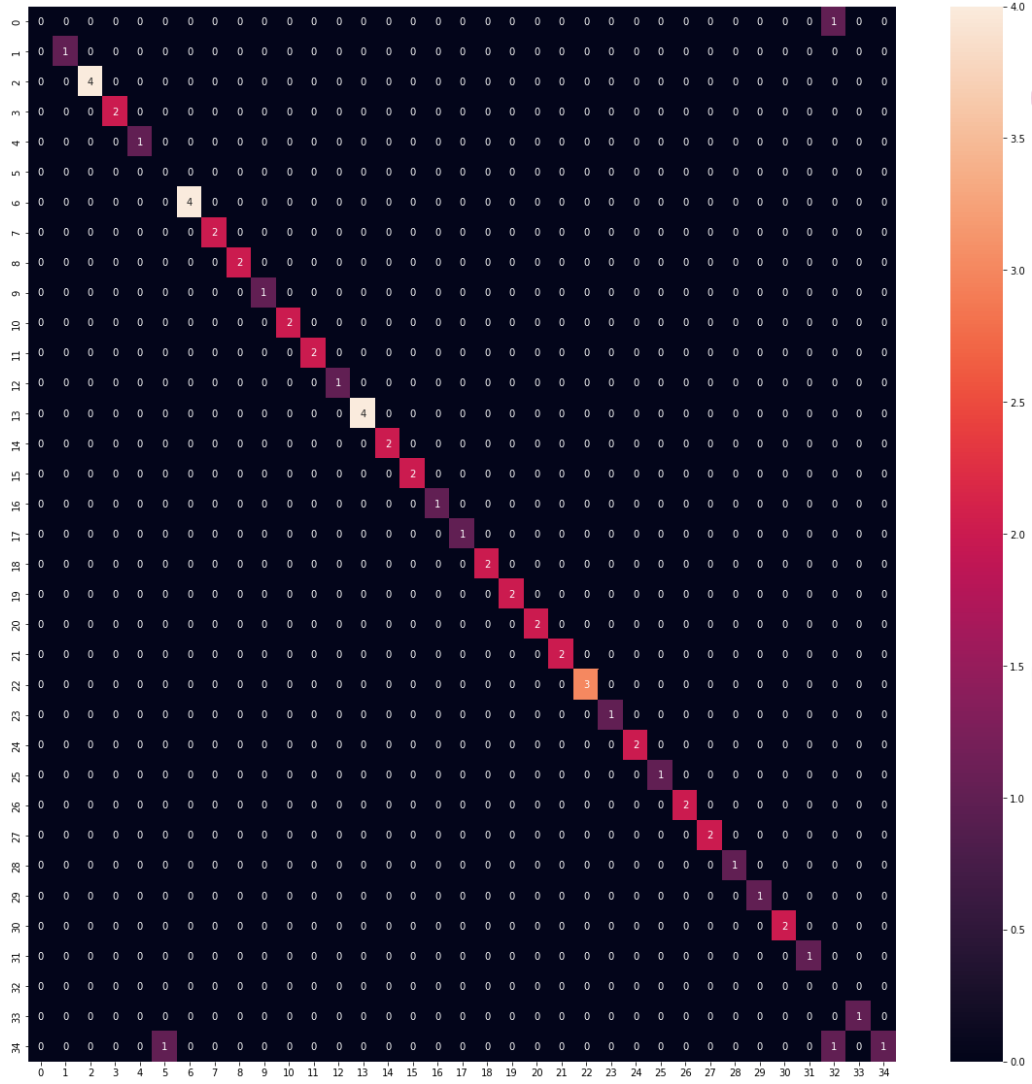
Accuracy Score

```
[ ] y_pred = rfc.predict(X_test)
    accuracy_score(y_test,y_pred)
```

```
0.9508196721311475
```

```
[ ] f1_score(y_test, y_pred, average='weighted')
```

```
0.9590163934426229
```



Experiment Results

Decision Tree

```
[ ]  from sklearn.tree import DecisionTreeClassifier

     dt = DecisionTreeClassifier(criterion='entropy', max_leaf_nodes=50 , random_state=42)
     dt.fit(X_train,y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_leaf_nodes=50, random_state=42)
```

```
[ ]  cross_val_score(dt,X_train,y_train,cv=3).mean()
```

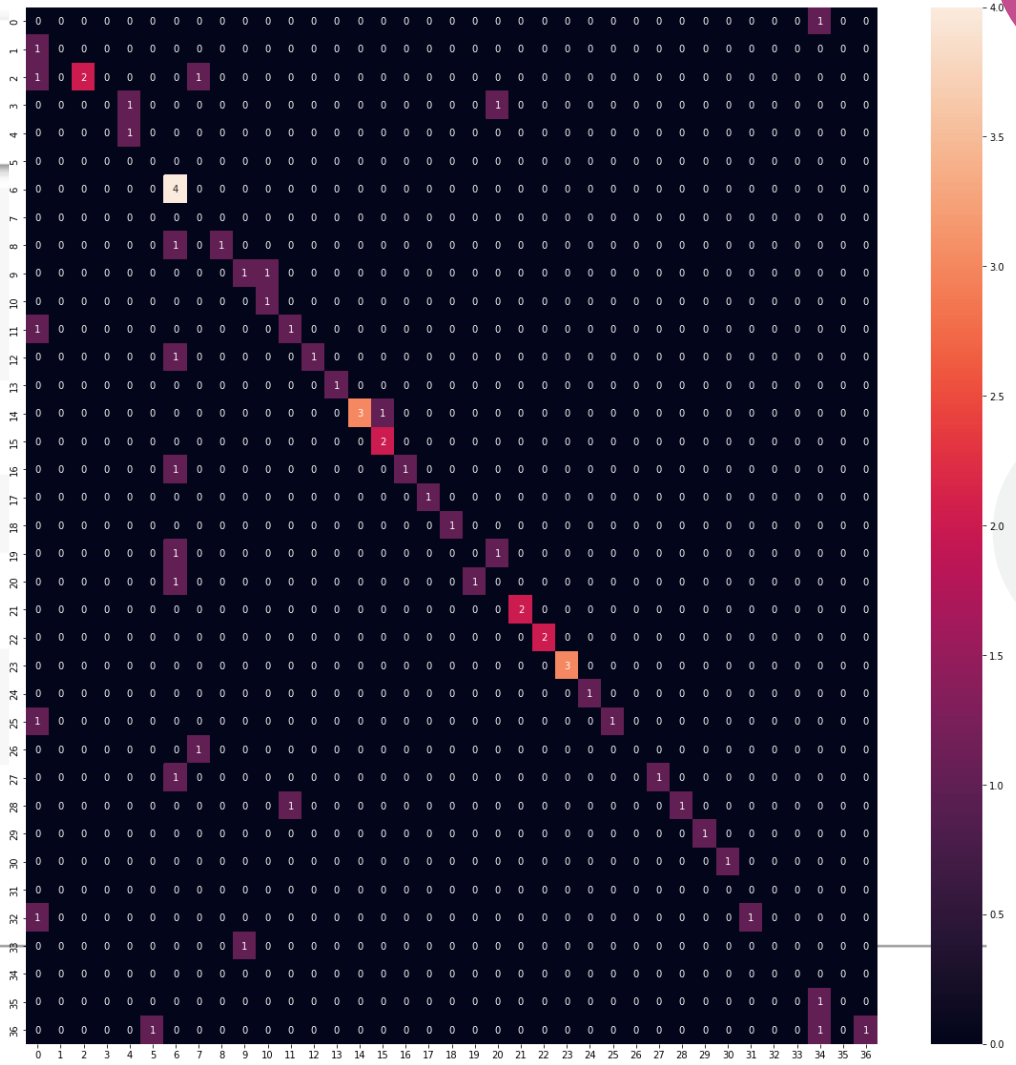
```
0.6049382716049383
```

```
[ ]  y_pred = dt.predict(X_test)
     accuracy_score(y_test,y_pred)
```

```
0.5737704918032787
```

```
[ ]  f1_score(y_test, y_pred, average='weighted')
```

```
0.5871194379391101
```



Experiment Results

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
nb.fit(X_train,y_train)
```

```
GaussianNB()
```

```
[ ] cross_val_score(nb,X_train,y_train,cv=3).mean()
```

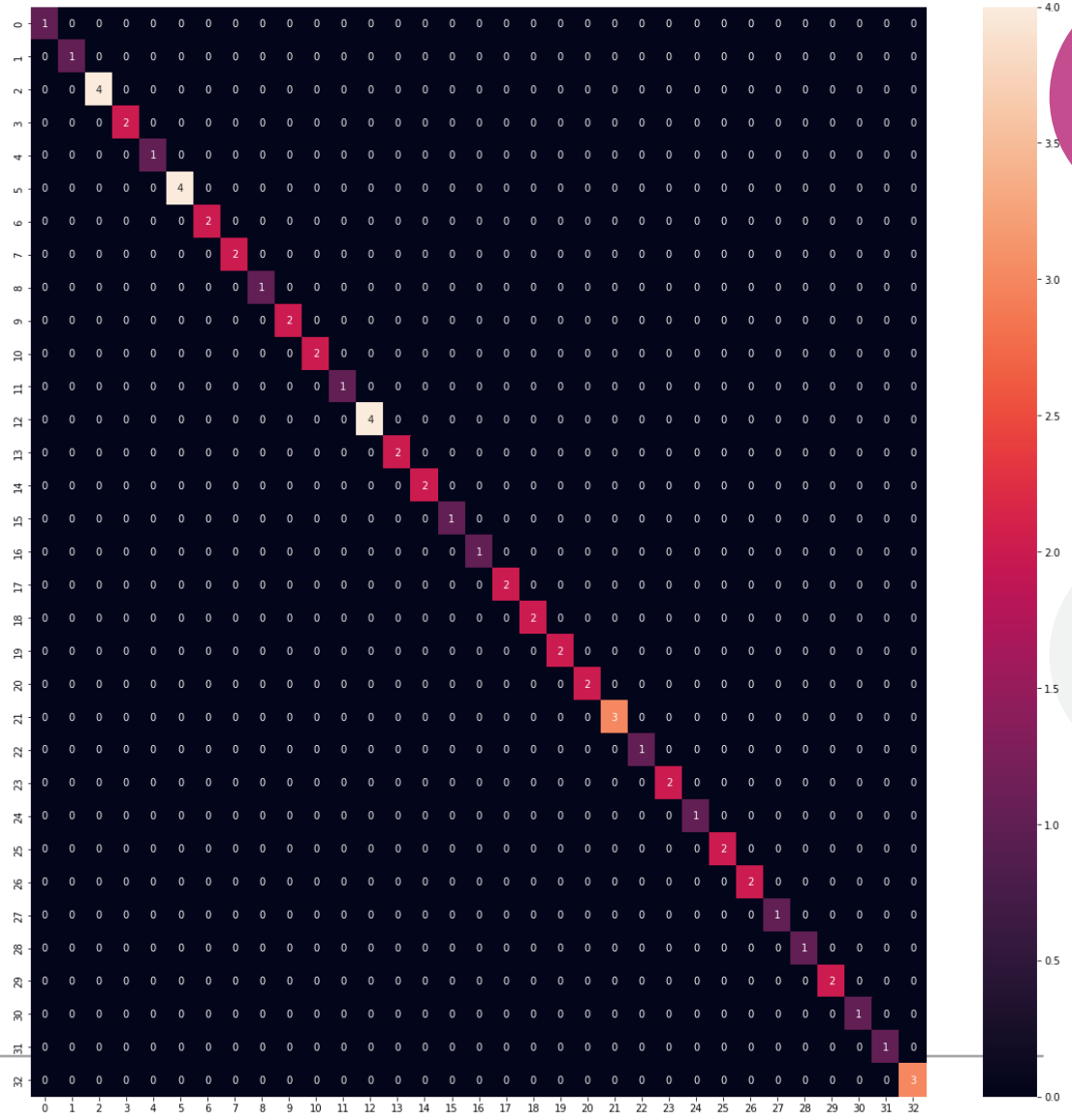
```
1.0
```

```
[ ] y_pred = nb.predict(X_test)
    accuracy_score(y_test,y_pred)
```

```
1.0
```

```
[ ] f1_score(y_test, y_pred, average='weighted')
```

```
1.0
```



Experiment Results

Ensemble

```
from sklearn.ensemble import VotingClassifier

vclf_soft = VotingClassifier(estimators=[('Naive Bayes', GaussianNB()),
                                         ('Random Forest Classifier', RandomForestClassifier(criterion='entropy',n_estimators=500, max_leaf_nodes=16, n_jobs=-1, random_state=42)),
                                         ('Decision Tree', DecisionTreeClassifier(criterion='entropy', max_leaf_nodes=50 , random_state=42))], voting='soft', n_jobs=-1)
```

```
[ ] vclf_soft.fit(X_train, y_train)
```

```
VotingClassifier(estimators=[('Naive Bayes', GaussianNB()),
                             ('Random Forest Classifier',
                              RandomForestClassifier(criterion='entropy',
                                                     max_leaf_nodes=16,
                                                     n_estimators=500,
                                                     n_jobs=-1,
                                                     random_state=42)),
                             ('Decision Tree',
                              DecisionTreeClassifier(criterion='entropy',
                                                     max_leaf_nodes=50,
                                                     random_state=42))],
                 n_jobs=-1, voting='soft')
```

```
[ ] cross_val_score(vclf_soft,X_train,y_train,cv=3).mean()
```

```
0.9958847736625515
```

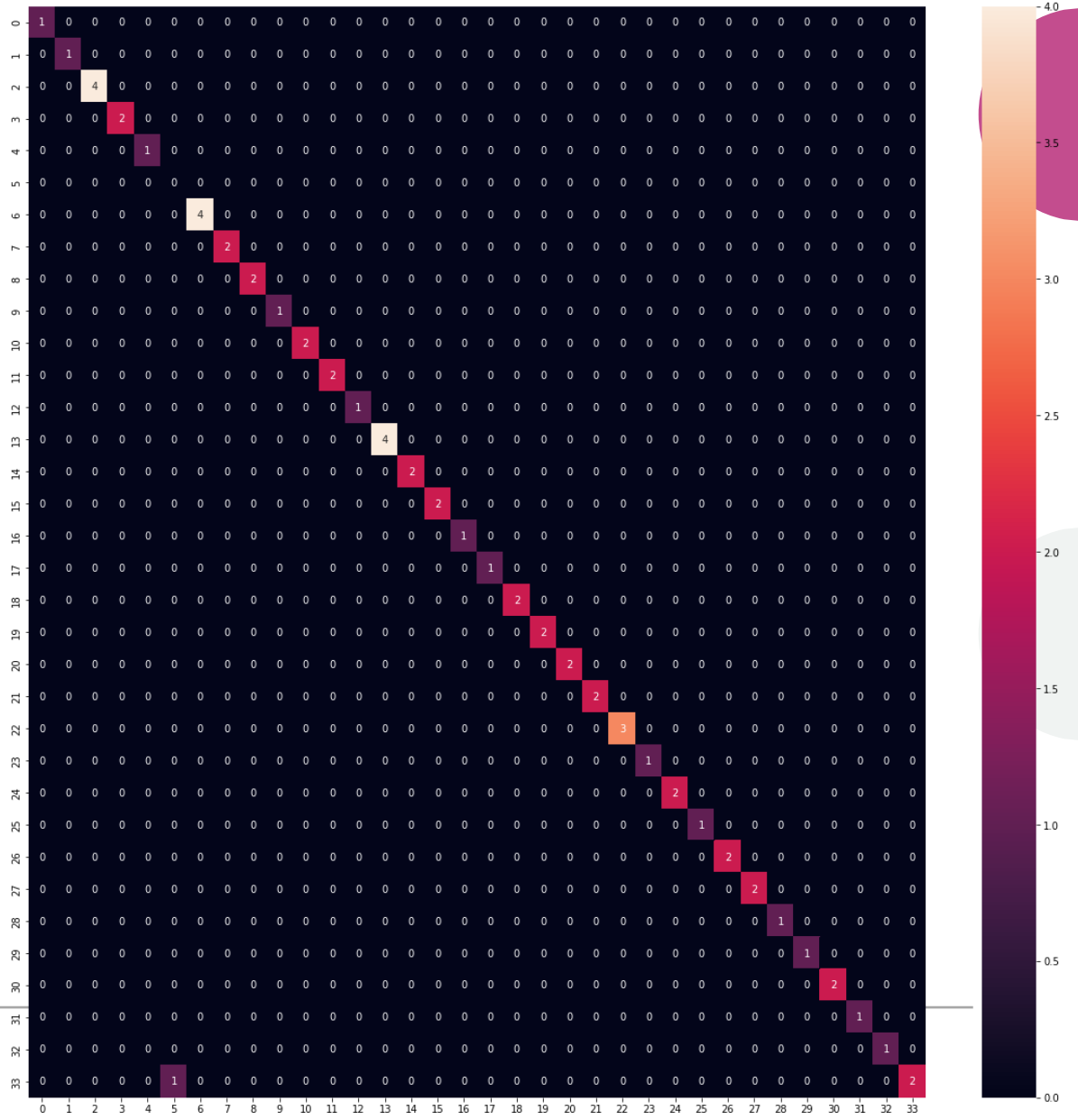
```
[ ] y_pred = vclf_soft.predict(X_test)
    accuracy_score(y_test,y_pred)
```

```
0.9836065573770492
```

```
[ ] f1_score(y_test, y_pred, average='weighted')
```

```
0.9901639344262295
```

Experiment Results



Ensemble’s Confusion Matrix

Experiment Results

PCA

```
[ ]  from sklearn.decomposition import PCA

     features = X

     pca = PCA(n_components=0.95)
     features_pca = pca.fit_transform(features,y)

     X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(features_pca,y, test_size=0.20,random_state=SEED, stratify=y)
```

With PCA we reduced the features from 131 to 46 while keeping 95% variance.

```
[ ]  print(X_train.shape,y_train.shape,X_test.shape,y_test.shape)

     (243, 131) (243,) (61, 131) (61,)
```

```
[ ]  print(X_train_pca.shape,y_train_pca.shape,X_test_pca.shape,y_test_pca.shape)

     (243, 46) (243,) (61, 46) (61,)
```

Experiment Results

Random Forest Classifier PCA

```
[ ]  rfclf_pca = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1,
     rfclf_pca.fit(X_train_pca, y_train_pca)
```

```
RandomForestClassifier(max_leaf_nodes=16, n_estimators=500, n_jobs=-1,
                       random_state=42)
```

```
[ ]  cross_val_score(rfclf_pca,X_train_pca,y_train_pca,cv=3).mean()
```

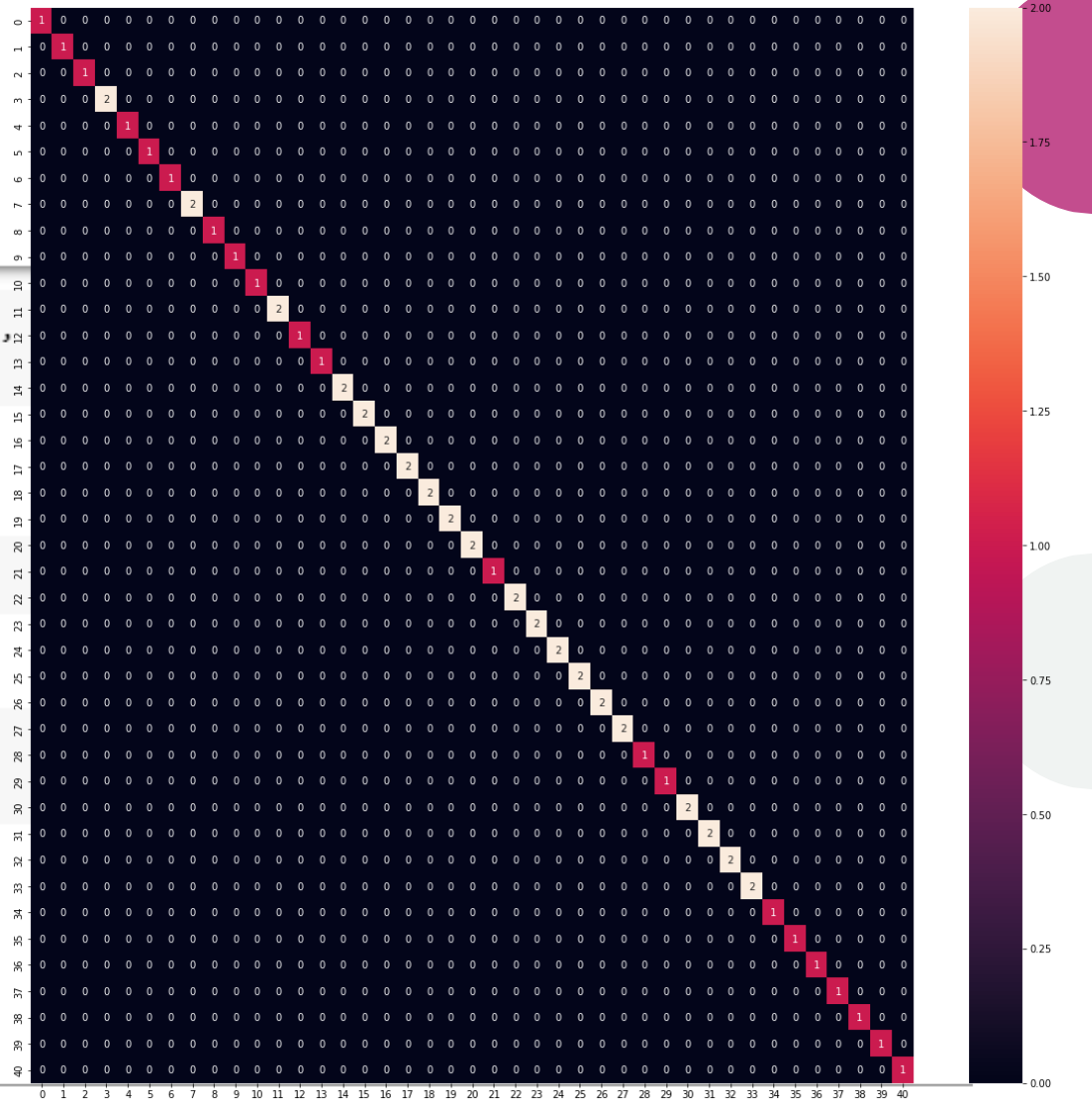
```
0.9794238683127571
```

```
[ ]  y_pred_pca = rfclf_pca.predict(X_test_pca)
     accuracy_score(y_test_pca,y_pred_pca)
```

```
1.0
```

```
[ ]  f1_score(y_test_pca, y_pred_pca, average='weighted')
```

```
1.0
```



Experiment Results

Decision Tree PCA

```
[ ]  dt_pca = DecisionTreeClassifier(criterion='entropy', max_leaf_nodes=50 , random_sta
     dt_pca.fit(X_train_pca,y_train_pca)
```

```
DecisionTreeClassifier(criterion='entropy', max_leaf_nodes=50, random_state=42)
```

```
[ ]  cross_val_score(dt_pca,X_train_pca,y_train_pca,cv=3).mean()
```

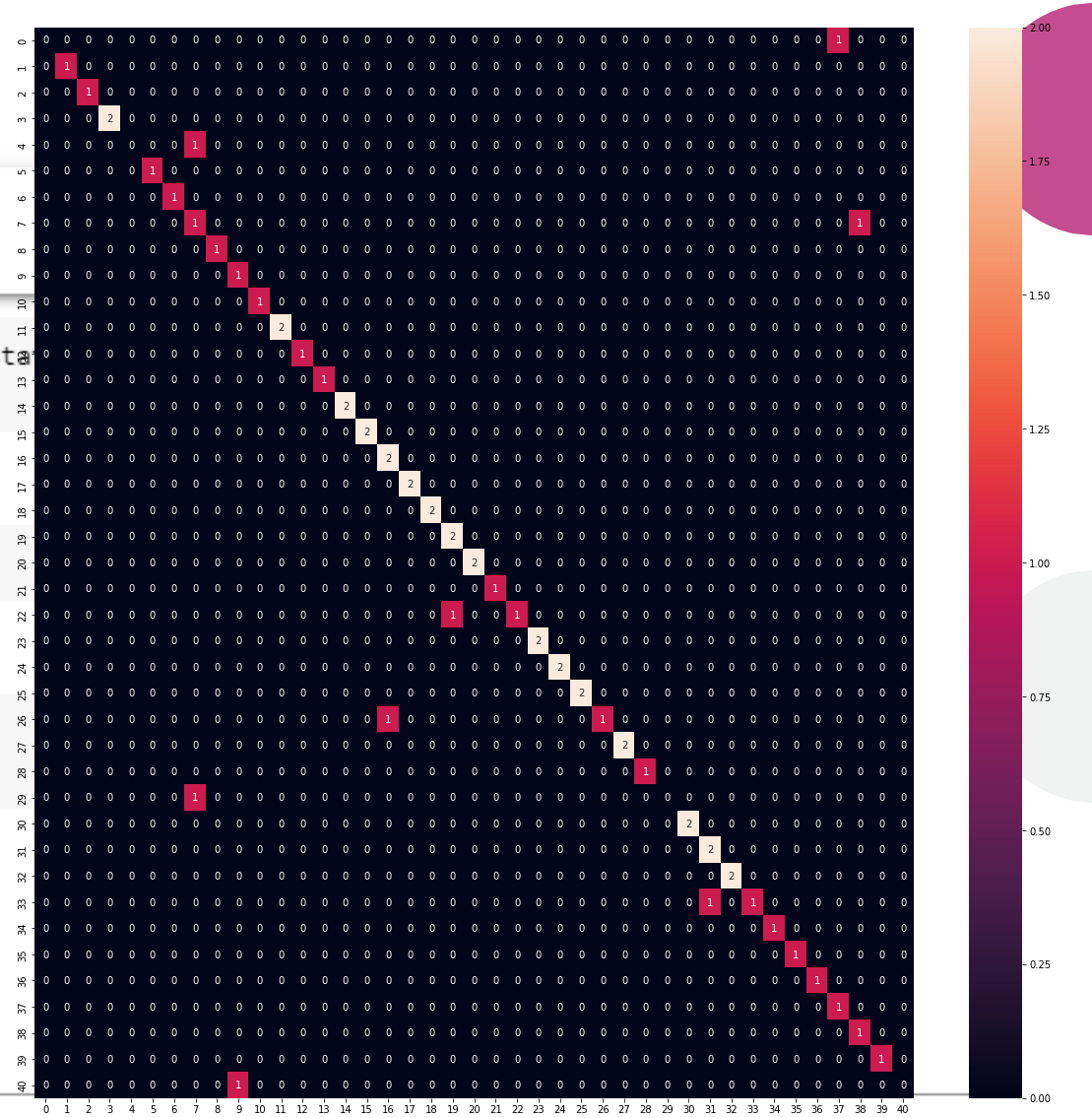
```
0.7818930041152262
```

```
[ ]  y_pred_pca = dt_pca.predict(X_test_pca)
     accuracy_score(y_test_pca,y_pred_pca)
```

```
0.8688524590163934
```

```
[ ]  f1_score(y_test_pca, y_pred_pca, average='weighted')
```

```
0.8459016393442622
```



Experiment Results

Naive Bayes PCA

```
[ ]  from sklearn.naive_bayes import GaussianNB

     nb_pca = GaussianNB()
     nb_pca.fit(X_train_pca,y_train_pca)
```

```
GaussianNB()
```

```
[ ]  cross_val_score(nb_pca,X_train_pca,y_train_pca,cv=3).mean()
```

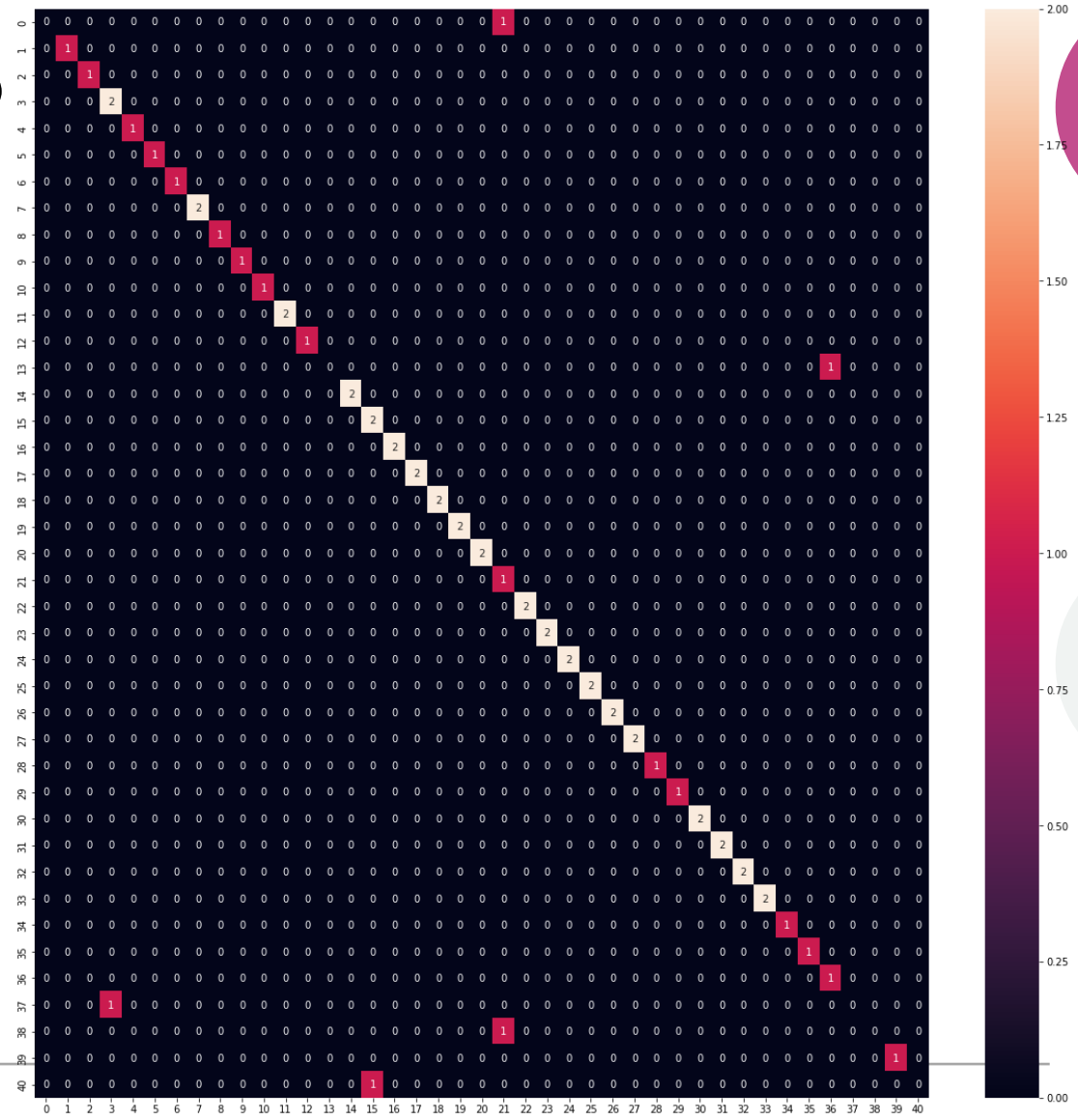
```
0.831275720164609
```

```
[ ]  y_pred_pca = nb_pca.predict(X_test_pca)
     accuracy_score(y_test_pca,y_pred_pca)
```

```
0.9180327868852459
```

```
[ ]  f1_score(y_test_pca, y_pred_pca, average='weighted')
```

```
0.891256830601093
```



Experiment Results

Ensemble PCA

```
[ ] vclf_soft_pca = VotingClassifier(estimators=[('Naive Bayes', GaussianNB()),
                                                 ('Random Forest Classifier', RandomForestClassifier(criterion='entropy',n_estimators=500, max_leaf_nodes=16, n_jobs=-1, random_state=42)),
                                                 ('Decision Tree', DecisionTreeClassifier(criterion='entropy', max_leaf_nodes=50 , random_state=42))], voting='soft', n_jobs=-1)
    vclf_soft_pca.fit(X_train_pca, y_train_pca)
```

```
VotingClassifier(estimators=[('Naive Bayes', GaussianNB()),
                             ('Random Forest Classifier',
                              RandomForestClassifier(criterion='entropy',
                                                     max_leaf_nodes=16,
                                                     n_estimators=500,
                                                     n_jobs=-1,
                                                     random_state=42)),
                             ('Decision Tree',
                              DecisionTreeClassifier(criterion='entropy',
                                                     max_leaf_nodes=50,
                                                     random_state=42))],
                 n_jobs=-1, voting='soft')
```

```
[ ] cross_val_score(vclf_soft_pca,X_train_pca,y_train_pca,cv=3).mean()
```

```
0.9218106995884773
```

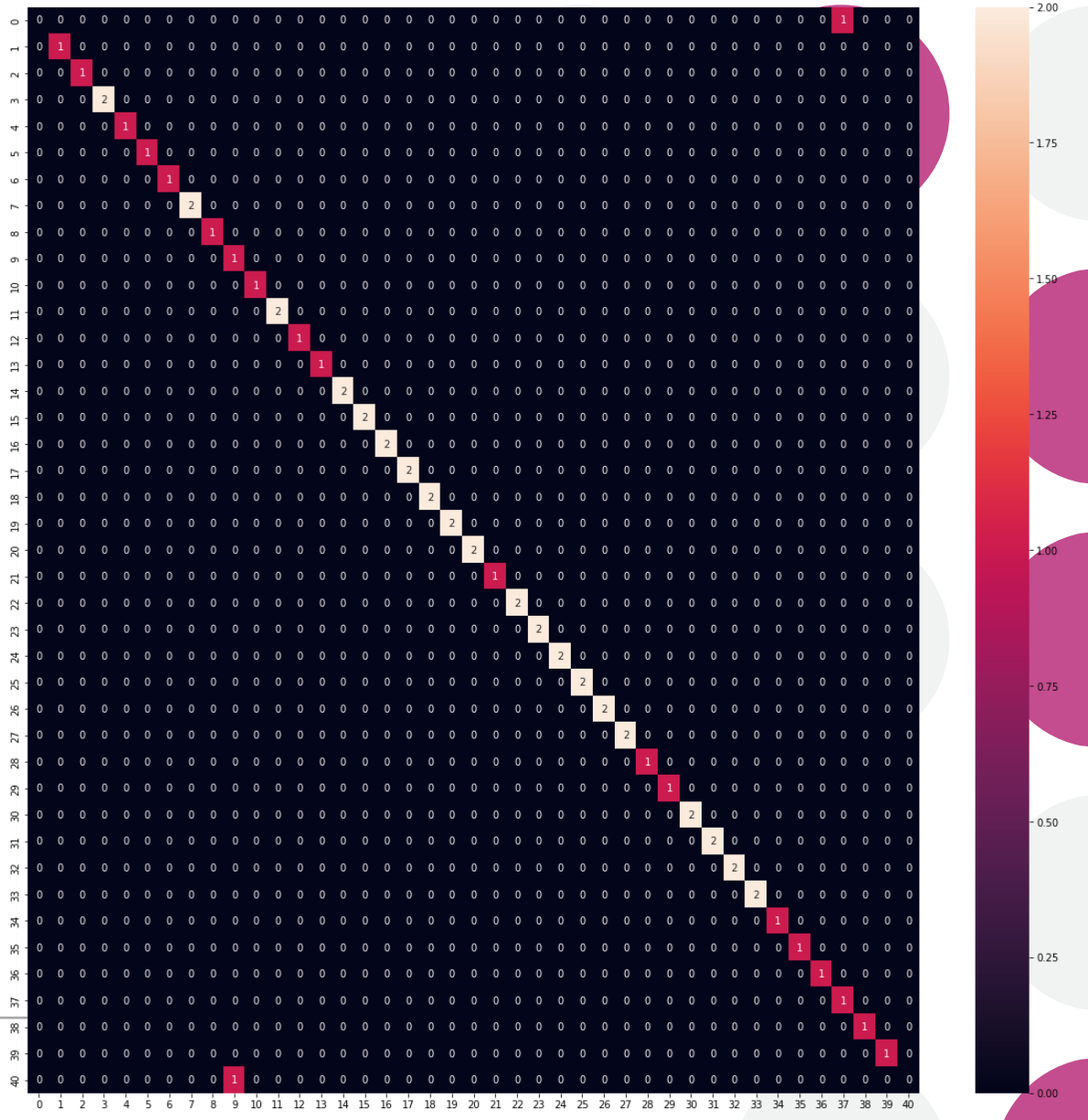
```
[ ] y_pred_pca = vclf_soft_pca.predict(X_test_pca)
    accuracy_score(y_test_pca,y_pred_pca)
```

```
0.9672131147540983
```

```
[ ] f1_score(y_test_pca, y_pred_pca, average='weighted')
```

```
0.9562841530054644
```

Experiment Results



Ensemble PCA’s Confusion Matrix

Key Findings

Best Performers:

- Random Forest Classifier with PCA 100% Accuracy Score
- Naïve Bayes with no PCA 100% Accuracy Score
- Ensemble with no PCA 98% Accuracy Score

Key Findings

- Machine learning could be used to change the way people have access to medical advisors. It could help medical professionals diagnose people faster. That would lead to medical services in less fortunate countries being less expensive.
- PCA is not going to improve all algorithms and datasets.
- PCA made Naïve bayes worse.