Sprint 4

Date	19.11.22
Team ID	PNT2022TMID53212
Project Title	Analytics for Hospitals' Health-Care Data
Team Members	Jaikishore R, Chitiprolu Prathyusha, Duvicksha U, Kapireddy Charitha

<u>ML Model-1</u>

1) **Naïve Bayes Model**

   In Bayes theorem, given a Hypothesis H and Evidence E, it states that the relation between the probability of Hypothesis P(H) before getting Evidence and probability of hypothesis after getting Evidence P(H|E) $P(H \mid E) = [\boldsymbol{P(E \mid H) / P(E)}]\, P(H)$ When we apply Bayes Theorem to our data it represents as follows. • P(H) is the prior probability of a patient's length of stay (LOS). • P(E) is the probability of a feature variable. • P(E|H) is the probability of a patient's LOS given that the features are true. • P(H|E) is the probability of the features given that patient's LOS is true. Model is trained using Gaussian Naïve Bayes classifier, partitioned train data is fed to the model in array format then the trained model is validated using validation data.

   MODELLING

   Naives Bayes Model

```
[38] from sklearn.naive_bayes import GaussianNB
     target = y_train.values
     features = X_train.values
     classifier_nb = GaussianNB()
     model_nb = classifier_nb.fit(features, target)
```

```
[39] prediction_nb = model_nb.predict(X_test)
     from sklearn.metrics import accuracy_score
     acc_score_nb = accuracy_score(prediction_nb,y_test)
     print("Acurracy:", acc_score_nb*100)

     Acurracy: 34.55439015199096
```

   This model gives an accuracy of 34.55% after validating

2) **XGBoost Model**

   Boosting is a sequential technique that works on the principle of an ensemble. At any instant T, the model outcomes are weighed based on the outcomes of the previous instant (T -1). It combines the set of weak learners and improves prediction accuracy. Tree ensemble is a set of classification and regression trees. Trees are grown one after another, and they try to reduce the misclassification rate. The final prediction score of the model is calculated by summing up each and individual score. Before feeding train data to the XGB Classifier model, booster parameters must be tuned. Tunning the model can prevent overfitting and can yield higher accuracy.

   In this XGBoost model, we have used the following parameters for tunning,

• learning_rate = 0.1 - step size shrinkage used to prevent overfitting. After each boosting step, we can directly get the weights of new features, and eta shrinks the feature weights to make the boosting process more conservative.
• max_depth = 4 – Maximum depth of the tree. This value describes the complexity of the model. Increasing its value results in overfitting.
• n_estimators = 800 – Number of gradient boosting trees or rounds. Each new tree attempts to model and correct for the errors made by the sequence of previous trees. Increasing the number of trees can yield higher accuracy but the model reaches a point of diminishing returns quickly.
• objective = ‘multi:softmax’ – this parameter sets XGBoost to do multiclass classification using the softmax objective because the target variable has 11 Levels.
• reg_alpha = 0.5 - L1 regularization term on weights. Increasing this value will make the model more conservative.
• reg_lambda = 1.5 - L2 regularization term on weights and is smoother than L1 regularization. Increasing this value will model more conservative.
• min_child_weight = 2 - Minimum sum of instance weight needed in a child.

XGBoost Model

```
[41] import xgboost
     classifier_xgb = xgboost.XGBClassifier(max_depth=4, learning_rate=0.1, n_estimators=800,
                                            objective='multi:softmax', reg_alpha=0.5, reg_lambda=1.5,
                                            booster='gbtree', n_jobs=4, min_child_weight=2, base_score= 0.75)

[42] model_xgb = classifier_xgb.fit(X_train, y_train)

     prediction_xgb = model_xgb.predict(X_test)
     acc_score_xgb = accuracy_score(prediction_xgb,y_test)
     print("Accuracy:", acc_score_xgb*100)

     Accuracy: 43.047355859816605
```

Once the model was trained and validated, it yields an accuracy score of 43.04%. This model nearly took 25 minutes to get trained but when compared to the Naïve Bayes model it gave an 8.5% improvement.

3) Neural Network Model

Neural Networks are built of simple elements called neurons, which take in a real value, multiply it by weight, and run it through a non-linear activation function. The process records one at a time and learns by comparing their classification of the record with the known actual classification of the record. The errors from the initial classification of the first record are fed back into the network and used to modify the network's algorithm for further iterations.In this neural network model, there are six dense layers, the final layer is an output layer with an activation function “SoftMax”. SoftMax is used here because each patient must be classified in one of the 11 levels in the Stay variable. In this model, increasing the number of neurons from each layer to the other layer, will increase the hypothetical space of the model and try to learn more patterns from the data. There are a total of 442,571 trainable parameters. Every layer is activated using “relu” activation function because it overcomes the vanishing gradient problem, allowing models to learn faster and perform better.

Analytics for hospital's Health-Care Data .ipynb

File Edit View Insert Runtime Tools Help All changes saved

\+ Code + Text

Neural Network Model

```
# Segregation of features and target variable
X = train.drop('Stay', axis =1)
y = train['Stay']
print(X.columns)
z = test.drop('Stay', axis = 1)
print(z.columns)

# Data Scaling
from sklearn import preprocessing
X_scale = preprocessing.scale(X)
X_scale.shape
```

```
Index(['case_id', 'Hospital_code', 'Hospital_type_code', 'City_Code_Hospital',
       'Hospital_region_code', 'Available Extra Rooms in Hospital',
       'Department', 'Ward_Type', 'Ward_Facility_Code', 'Bed Grade',
       'patientid', 'City_Code_Patient', 'Type of Admission',
       'Severity of Illness', 'Visitors with Patient', 'Age',
       'Admission_Deposit', 'count_id_patient',
       'count_id_patient_hospitalCode', 'count_id_patient_wardfacilityCode'],
      dtype='object')
Index(['case_id', 'Hospital_code', 'Hospital_type_code', 'City_Code_Hospital',
       'Hospital_region_code', 'Available Extra Rooms in Hospital',
       'Department', 'Ward_Type', 'Ward_Facility_Code', 'Bed Grade',
       'patientid', 'City_Code_Patient', 'Type of Admission',
       'Severity of Illness', 'Visitors with Patient', 'Age',
       'Admission_Deposit', 'count_id_patient',
       'count_id_patient_hospitalCode', 'count_id_patient_wardfacilityCode'],
      dtype='object')
(318438, 20)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scale, y, test_size =0.20, random_state =100)
```

```
[31] import keras
     from keras.models import Sequential
     from keras.layers import Dense
     import tensorflow as tf
```

```
[32] from keras.utils import to_categorical
     #Sparse Matrix
     a = to_categorical(y_train)
     b = to_categorical(y_test)
```

```
[33] model = Sequential()
     model.add(Dense(64, activation='relu', input_shape = (20,)))
     model.add(Dense(128, activation='relu'))
     model.add(Dense(256, activation='relu'))
     model.add(Dense(512, activation='relu'))
     model.add(Dense(512, activation='relu'))
     model.add(Dense(11, activation='softmax'))
```

```
[34] model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                1344

 dense_1 (Dense)             (None, 128)               8320

 dense_2 (Dense)             (None, 256)               33024

 dense_3 (Dense)             (None, 512)               131584

 dense_4 (Dense)             (None, 512)               262656

 dense_5 (Dense)             (None, 11)                5643

=================================================================
Total params: 442,571
Trainable params: 442,571
Non-trainable params: 0
_________________________________________________________________
```

```
[35] model.compile(optimizer= 'SGD',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])
```

```
[36] callbacks = [tf.keras.callbacks.TensorBoard("logs_keras")]
     model.fit(X_train, a, epochs=20, callbacks=callbacks, validation_split = 0.2)
```

```
Epoch 1/20
6369/6369 [==============================] - 60s 9ms/step - loss: 1.6523 - accuracy: 0.3712 - val_loss: 1.5908 - val_accuracy: 0.3940
Epoch 2/20
6369/6369 [==============================] - 62s 10ms/step - loss: 1.5681 - accuracy: 0.3997 - val_loss: 1.5665 - val_accuracy: 0.4012
Epoch 3/20
6369/6369 [==============================] - 60s 9ms/step - loss: 1.5480 - accuracy: 0.4068 - val_loss: 1.5515 - val_accuracy: 0.4044
Epoch 4/20
6369/6369 [==============================] - 57s 9ms/step - loss: 1.5346 - accuracy: 0.4109 - val_loss: 1.5367 - val_accuracy: 0.4118
Epoch 5/20
6369/6369 [==============================] - 62s 10ms/step - loss: 1.5242 - accuracy: 0.4149 - val_loss: 1.5344 - val_accuracy: 0.4119
Epoch 6/20
6369/6369 [==============================] - 57s 9ms/step - loss: 1.5172 - accuracy: 0.4175 - val_loss: 1.5288 - val_accuracy: 0.4137
Epoch 7/20
6369/6369 [==============================] - 57s 9ms/step - loss: 1.5112 - accuracy: 0.4191 - val_loss: 1.5218 - val_accuracy: 0.4168
Epoch 8/20
6369/6369 [==============================] - 56s 9ms/step - loss: 1.5066 - accuracy: 0.4205 - val_loss: 1.5191 - val_accuracy: 0.4184
Epoch 9/20
6369/6369 [==============================] - 56s 9ms/step - loss: 1.5021 - accuracy: 0.4227 - val_loss: 1.5175 - val_accuracy: 0.4184
Epoch 10/20
6369/6369 [==============================] - 56s 9ms/step - loss: 1.4980 - accuracy: 0.4237 - val_loss: 1.5195 - val_accuracy: 0.4166
```

```
[36] Epoch 10/20
6369/6369 [==============================] - 56s 9ms/step - loss: 1.4980 - accuracy: 0.4237 - val_loss: 1.5195 - val_accuracy: 0.4166
Epoch 11/20
6369/6369 [==============================] - 56s 9ms/step - loss: 1.4945 - accuracy: 0.4246 - val_loss: 1.5139 - val_accuracy: 0.4182
Epoch 12/20
6369/6369 [==============================] - 56s 9ms/step - loss: 1.4916 - accuracy: 0.4257 - val_loss: 1.5136 - val_accuracy: 0.4175
Epoch 13/20
6369/6369 [==============================] - 55s 9ms/step - loss: 1.4885 - accuracy: 0.4272 - val_loss: 1.5107 - val_accuracy: 0.4195
Epoch 14/20
6369/6369 [==============================] - 56s 9ms/step - loss: 1.4853 - accuracy: 0.4280 - val_loss: 1.5079 - val_accuracy: 0.4220
Epoch 15/20
6369/6369 [==============================] - 55s 9ms/step - loss: 1.4826 - accuracy: 0.4293 - val_loss: 1.5085 - val_accuracy: 0.4216
Epoch 16/20
6369/6369 [==============================] - 61s 10ms/step - loss: 1.4800 - accuracy: 0.4303 - val_loss: 1.5052 - val_accuracy: 0.4223
Epoch 17/20
6369/6369 [==============================] - 57s 9ms/step - loss: 1.4772 - accuracy: 0.4312 - val_loss: 1.5045 - val_accuracy: 0.4215
Epoch 18/20
6369/6369 [==============================] - 56s 9ms/step - loss: 1.4750 - accuracy: 0.4309 - val_loss: 1.5043 - val_accuracy: 0.4218
Epoch 19/20
6369/6369 [==============================] - 56s 9ms/step - loss: 1.4728 - accuracy: 0.4325 - val_loss: 1.5076 - val_accuracy: 0.4221
Epoch 20/20
6369/6369 [==============================] - 56s 9ms/step - loss: 1.4704 - accuracy: 0.4340 - val_loss: 1.5023 - val_accuracy: 0.4218
<keras.callbacks.History at 0x7f483d5104d0>
```

```
[39] # Retraining the model with 4 epochs
     model.fit(X_train, a, epochs=4, validation_split = 0.2)
     print("\n Model Evaluation")
     model.evaluate(X_test,b)
```

```
Epoch 1/4
6369/6369 [==============================] - 56s 9ms/step - loss: 1.4674 - accuracy: 0.4349 - val_loss: 1.5081 - val_accuracy: 0.4193
Epoch 2/4
6369/6369 [==============================] - 56s 9ms/step - loss: 1.4655 - accuracy: 0.4350 - val_loss: 1.5049 - val_accuracy: 0.4215
Epoch 3/4
6369/6369 [==============================] - 62s 10ms/step - loss: 1.4630 - accuracy: 0.4358 - val_loss: 1.5097 - val_accuracy: 0.4223
Epoch 4/4
6369/6369 [==============================] - 58s 9ms/step - loss: 1.4604 - accuracy: 0.4375 - val_loss: 1.5093 - val_accuracy: 0.4160

 Model Evaluation
1991/1991 [==============================] - 8s 4ms/step - loss: 1.5078 - accuracy: 0.4179
[1.5078129768371582, 0.41794371604919434]
```

Finally, evaluating the model with a test set yields an accuracy score of 41.79%. Neural Networks supposedly performs better than any other models. But because of the smaller dataset, it was not able to learn more accurately than the XGBoost model. It nearly took 20 minutes to train the model.

Predictions

Predictions

```
[39] # Naive Bayes
pred_nb = classifier_nb.predict(test1.iloc[:,1:])
result_nb = pd.DataFrame(pred_nb, columns=['Stay'])
result_nb['case_id'] = test1['case_id']
result_nb = result_nb[['case_id', 'Stay']]
```

```
result_nb['Stay'] = result_nb['Stay'].replace(
    {0:'0-10',
     1: '11-20',
     2: '21-30',
     3:'31-40',
     4: '41-50',
     5: '51-60',
     6: '61-70',
     7: '71-80',
     8: '81-90',
     9: '91-100',
     10: 'More than 100 Days'})
result_nb.head()
```

	case_id	Stay
0	318439	21-30
1	318440	51-60
2	318441	21-30
3	318442	21-30
4	318443	31-40

```
# XGBoost
pred_xgb = classifier_xgb.predict(test1.iloc[:,1:],validate_features=False)
result_xgb = pd.DataFrame(pred_xgb, columns=['Stay'])
result_xgb['case_id'] = test1['case_id']
result_xgb = result_xgb[['case_id', 'Stay']]
```

```
[44] result_xgb['Stay'] = result_xgb['Stay'].replace({0:'0-10', 1: '11-20', 2: '21-30'
result_xgb.head()
```

	case_id	Stay
0	318439	0-10
1	318440	51-60
2	318441	21-30
3	318442	21-30
4	318443	51-60

```
# Neural Network
test_scale = preprocessing.scale(z)
test_scale.shape
```

```
(137057, 20)
```

```
[48] pred1 = model.predict(test_scale)
     pred=np.argmax(pred1,axis=1)
     pred
```

```
4284/4284 [==============================] - 13s 3ms/step
array([0, 5, 2, ..., 2, 2, 5])
```

```
[49] result_nn = pd.DataFrame(pred, columns=['Stay'])
     result_nn['case_id'] = test['case_id']
     result_nn = result_nn[['case_id', 'Stay']]
```

```
[50] result_nn['Stay'] = result_nn['Stay'].replace({0:'0-10', 1: '11-20', 2: '21-30', 3:'
     result_nn.head()
```

	case_id	Stay
0	318439	0-10
1	318440	51-60
2	318441	21-30
3	318442	21-30

<u>Results</u>

RESULTS

```
[51] # Naive Bayes
     print(result_nb.groupby('Stay')['case_id'].nunique())
```

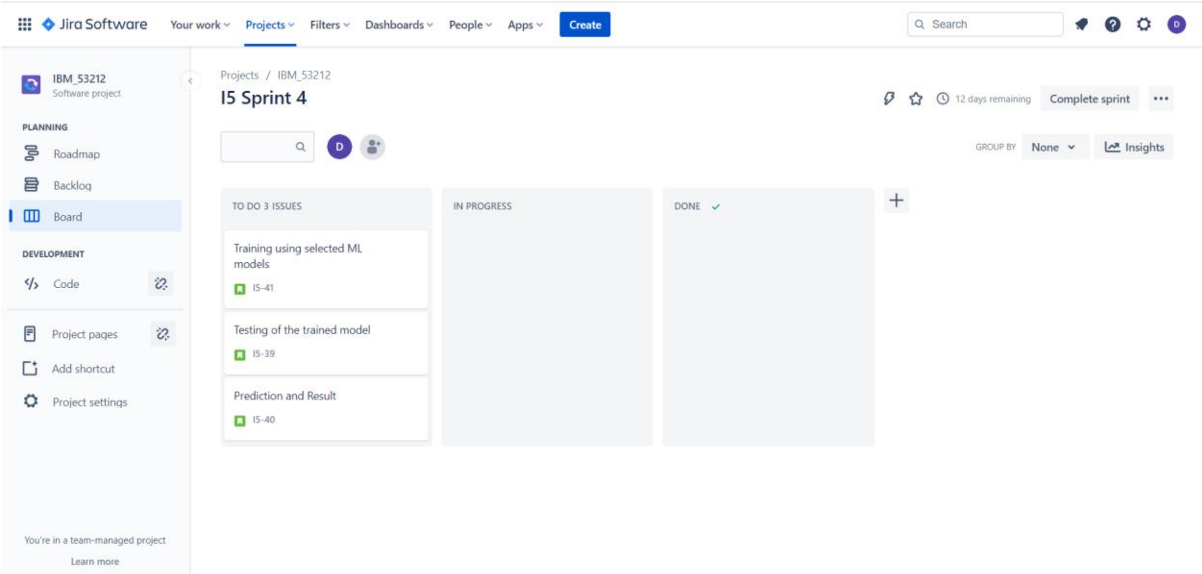
```
Stay
0-10                  2598
11-20                26827
21-30                72206
31-40                15639
41-50                  469
51-60                13651
61-70                   92
71-80                  955
81-90                  296
91-100                   2
More than 100 Days    4322
Name: case_id, dtype: int64
```

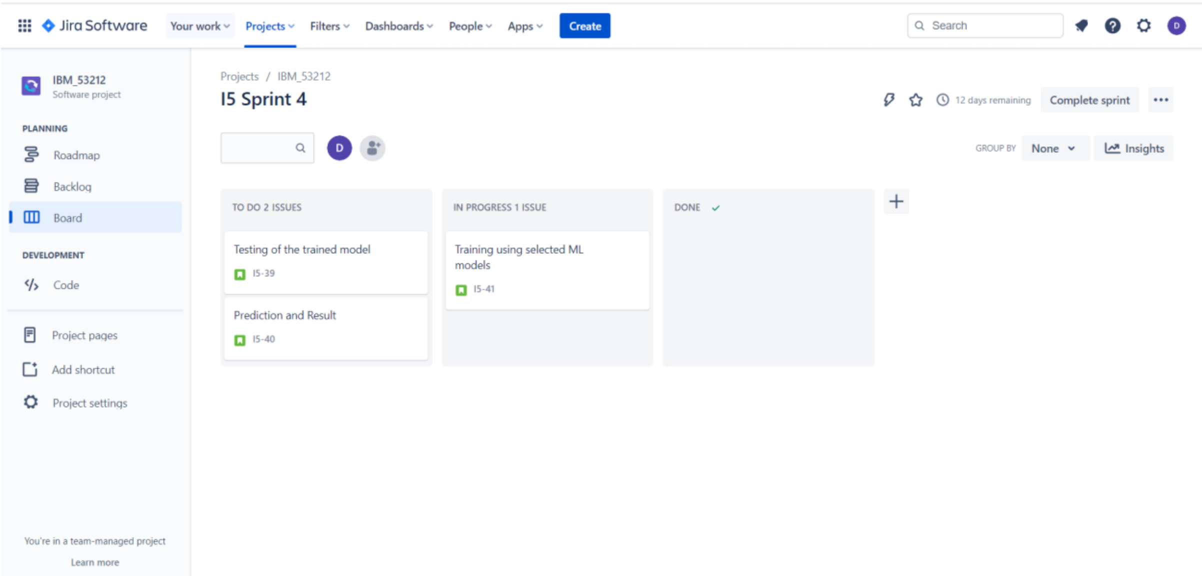
```
[52] # XGBoost
     print(result_xgb.groupby('Stay')['case_id'].nunique())

     Stay
     0-10                   4373
     11-20                 39337
     21-30                 58261
     31-40                 12100
     41-50                    61
     51-60                 19217
     61-70                    16
     71-80                   302
     81-90                  1099
     91-100                   78
     More than 100 Days     2213
     Name: case_id, dtype: int64
```

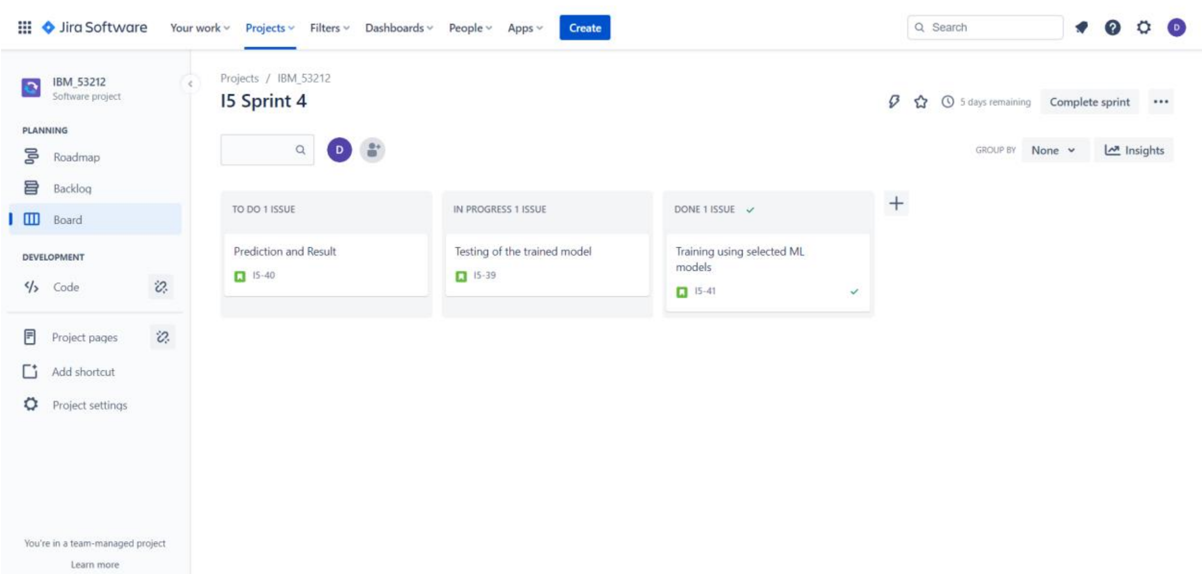
```
[53] # Neural Networks
     print(result_nn.groupby('Stay')['case_id'].nunique())

     Stay
     0-10                   4940
     11-20                 26115
     21-30                 69939
     31-40                  8862
     41-50                    57
     51-60                 22697
     71-80                   168
     81-90                  1066
     More than 100 Days     3213
     Name: case_id, dtype: int64
```

In the Naive Bayes model, patients are more likely to be misclassified. This model is biased towards the duration of 21-30 days, it has classified 72,206 patients for this level. Whereas the other two models XGBoost and Neural Networks are predicting mostly similar Length of Stay for the patient Examining these predictions, many of the patients are staying in the hospital for 21-30 days and very few people are staying for 61-70 days. As far as the distribution of Length of Stay is concerned, 13% of the patients are discharged from the hospital within 20 days and 1% of the overall patients are staying in the hospital for more than 60 days.

JIRA Sprint 4 Tracking







Sprint 4 Completed Successfully