Approach for Problem 1

We are using machine learning for solving the problem. Dynamic Price is assumed to be base price plus variable price. Base Price depend on the source and destination cities (fuel surcharge and crew salary) and ‘class’. Variable Price depends can depend on day of the week, age of the individual (concessions), time of booking(sale offers).

Generating feature vectors:

We have tried different approaches for representing the given data in feature vectors.

‘Date of Birth’ data is used to get age of the individuals. We tried representing age data as one hot vector(age\_vec) of four dimensions corresponding the age groups [0-12,12-25,25-50,50>]. Age is then directly used as a number in the feature vector. ‘From’ and ‘To’ cities are used as one hot coded vector(city\_vec) of dimension equal to number of cities with 2 for the source and 1 for the destination city. ‘Flight date’ data is used as hot coded vector of 7 dimension corresponding to the day of the week. Day of the week is then future categorized in two weekday and weekend using a flag. Also ‘Flight date’ data is used for determining no. of days difference between day of booking and flight date(duration). This difference is also used as a feature in the feature vector. ‘Flight time’ data is converted to minutes(flight\_time) before appending to the feature vector. ‘Class’ data is two dimensional hot coded vector. ‘Fare’ data is used to learn the dynamic pricing. It is not a part of the feature vector.

Training using different ML methods:

We have used various types of methods like Logistic Regression, Support Vector Regression, Kernel Ridge Regression, MultiLayer Perceptron and Feed Forward Neural Network with activation layer using the sklearn and keras libraries.

1. Using feature vector: [age\_vec, city\_vec, duration, flight\_time]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Logistic Regression | Support Vector Regression | Kernel Ridge Regression | MultiLayer Perceptron |
| Accuracy | 31.64% | 35.13% | 60.65% | 71.68% |

We found that when the MLPRegressor perform best in this scenario.

It is also worth noting that there were some prediction fares which are negative in this case. So, we planned to use Feed Forward Neural Network with activation layer of softmax.

2. Using feature vector: [age\_vec, city\_vec, flight\_claas, weekday\_vec, duration, flight\_time]

(here weekday\_vec is one-hot coded)

|  |  |  |
| --- | --- | --- |
| Method | MultiLayer Perceptron | Feed Forward Neural Network |
| Accuracy | 75.05% | 81.02% |

3. Using feature vector: [age\_vec, city\_vec, flight\_claas, weekday\_vec, duration, flight\_time] (here weekday\_vec is of size 2)

[age\_vec, city\_vec, flight\_claas, weekday\_vec, duration, flight\_time] w/ weekday\_vec of size 2

(64, 16, 16, ) ~y1 acc ~ 0.80220998

keras:

classifier.add(Dense(output\_dim = 64, init = 'uniform', activation = 'relu', input\_dim = 17))

# Adding the intermediate hidden layers

classifier.add(Dense(output\_dim = 16, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

# Adding the output layer

classifier.add(Dense(output\_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling Neural Network

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

# Fitting our model

classifier.fit(X\_train\_mat[:split\_ind], Y\_train\_mat[:split\_ind], batch\_size = 50, epochs = 1000)

accuracy@1000 epochs, 50 batch\_size ~85.13

accuracy@10000 epochs, 100 batch\_size ~78.21

# Adding the input layer and the first hidden layer

classifier.add(Dense(output\_dim = 64, init = 'uniform', activation = 'relu', input\_dim = 17))

# Adding the intermediate hidden layers

classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 16, init = 'uniform', activation = 'relu'))

# classifier.add(Dense(output\_dim = 16, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

# classifier.add(Dropout(0.5))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

# Adding the output layer

classifier.add(Dense(output\_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling Neural Network

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

# Fitting our model

classifier.fit(X\_train\_mat[:split\_ind], Y\_train\_mat[:split\_ind], batch\_size = 100, epochs = 1000)

accuracy@1000 epochs, 100 batch\_size ~85.716

# Adding the input layer and the first hidden layer

classifier.add(Dense(output\_dim = 128, init = 'uniform', activation = 'relu', input\_dim = 17))

# Adding the intermediate hidden layers

classifier.add(Dense(output\_dim = 64, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu'))

# classifier.add(Dense(output\_dim = 16, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

# classifier.add(Dropout(0.5))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

# Adding the output layer

classifier.add(Dense(output\_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling Neural Network

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

# Fitting our model

classifier.fit(X\_train\_mat[:split\_ind], Y\_train\_mat[:split\_ind], batch\_size = 100, epochs = 1000)

accuracy ~77.5

# Adding the input layer and the first hidden layer

classifier.add(Dense(output\_dim = 64, init = 'uniform', activation = 'relu', input\_dim = 17))

# Adding the intermediate hidden layers

classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 16, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 16, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

# classifier.add(Dropout(0.5))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

# Adding the output layer

classifier.add(Dense(output\_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling Neural Network

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

# Fitting our model

classifier.fit(X\_train\_mat[:split\_ind], Y\_train\_mat[:split\_ind], batch\_size = 50, epochs = 1000)

accuracy ~84.079

# Adding the input layer and the first hidden layer

classifier.add(Dense(output\_dim = 64, init = 'uniform', activation = 'relu', input\_dim = 17))

# Adding the intermediate hidden layers

# classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu'))

# classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 128, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 16, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

# classifier.add(Dropout(0.5))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

# Adding the output layer

classifier.add(Dense(output\_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling Neural Network

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

# Fitting our model

classifier.fit(X\_train\_mat[:split\_ind], Y\_train\_mat[:split\_ind], batch\_size = 100, epochs = 1000)

accuracy ~83.43

classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu', input\_dim = 17))

# Adding the intermediate hidden layers

# classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu'))

# classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 16, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

# classifier.add(Dropout(0.5))

classifier.add(Dense(output\_dim = 4, init = 'uniform', activation = 'relu'))

# Adding the output layer

classifier.add(Dense(output\_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling Neural Network

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

# Fitting our model

classifier.fit(X\_train\_mat[:split\_ind], Y\_train\_mat[:split\_ind], batch\_size = 100, epochs = 1000)

accuracy ~86.71

classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu', input\_dim = 17))

# Adding the intermediate hidden layers

# classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu'))

# classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 4, init = 'uniform', activation = 'relu'))

# classifier.add(Dropout(0.5))

classifier.add(Dense(output\_dim = 4, init = 'uniform', activation = 'relu'))

# Adding the output layer

classifier.add(Dense(output\_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling Neural Network

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

# Fitting our model

classifier.fit(X\_train\_mat[:split\_ind], Y\_train\_mat[:split\_ind], batch\_size = 100, epochs = 1000)

accuracy ~87.02

classifier.add(Dense(output\_dim = 64, init = 'uniform', activation = 'relu', input\_dim = 17))

# Adding the intermediate hidden layers

# classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu'))

# classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 4, init = 'uniform', activation = 'relu'))

# classifier.add(Dropout(0.5))

classifier.add(Dense(output\_dim = 4, init = 'uniform', activation = 'relu'))

# Adding the output layer

classifier.add(Dense(output\_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling Neural Network

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

# Fitting our model

classifier.fit(X\_train\_mat[:split\_ind], Y\_train\_mat[:split\_ind], batch\_size = 100, epochs = 1000)

accuracy ~86.00

++++++++++++++++++++++++++++++++++++++++

[age, city\_vec, flight\_class, weekday\_vec, duration, flight\_time] w/ weekday\_vec of size 2

classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu', input\_dim = 14))

# Adding the intermediate hidden layers

# classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu'))

# classifier.add(Dense(output\_dim = 32, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 8, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output\_dim = 4, init = 'uniform', activation = 'relu'))

# classifier.add(Dropout(0.5))

classifier.add(Dense(output\_dim = 4, init = 'uniform', activation = 'relu'))

# Adding the output layer

classifier.add(Dense(output\_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling Neural Network

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

# Fitting our model

classifier.fit(X\_train\_mat[:split\_ind], Y\_train\_mat[:split\_ind], batch\_size = 100, epochs = 1000)

accuracy ~88.17

++++++++++++++++++++++++++++++++++++++++

without age:

accuracy 83.4

++++++++++++=

age c4 from 50+

acc ~86.67