PREDICTION OF NEW USER'S DESTINATION ON AIRBNB DATASET

Ayush Jain, Shweta Pathak, Ramya Sri Gujjari

{jain.ayu, pathak.sh, gujjari.r} @husky.neu.edu

INFO 7390, Spring 2018, Northeastern University

1. Abstract

Airbnb is a trusted community marketplace for people to list, discover and book unique accommodations around the world. For Airbnb, the enormous user data that they have been inherently maintaining is a mine of gold. Performing data mining based on customer's historical behavior statistics can reveal a lot of hidden information that wasn't paid attention to in the past. Maneuvering this data can help to effectively enhance user experience and increase the total booking of Airbnb.

In this paper, we develop a model to predict the destination city of a new user on Airbnb based on their demographics. The original dataset was taken from Kaggle competition and after performing comprehensive analysis, we enhanced it by adding new predictors related to the user's demographics and existing customer's data. Since this is a classification problem, we deployed a number of methodologies like Multinomial Naive Bayes Classifier (MNB), Multilayer perceptron classifier (MLP), Random Forest Classifier (RFC), Support Vector Machine Algorithm (SVM) and Recurrent Neural Networks (RNN) and tested the dataset on each of them. After comparing each of these models and their accuracies with Kaggle's competitions models, we found that for Random Forest Classifier, we are able to achieve accuracy of 68% which is better than Kaggle's 66% after adding additional features.

This report contains an introduction to the problem and comprehensive analysis on the dataset followed by feature engineering. Then we applied several classification models and after comparing each of their results, we proposed the best model which is RNN with an accuracy of *99.7%.*

2. Introduction

Airbnb is an American company which operates an online marketplace and hospitality service for people to lease or rent short-term lodging including holiday
cottages, apartments, homestays, hostel beds, or hotel rooms, to participate in or facilitate experiences related to tourism such as walking tours, and to make reservations at restaurants. [1] By predicting the destination of a new user accurately, Airbnb can enhance user experience and fasten the average time taken for first bookings. Since Airbnb is a common and a famous portal that is used on a daily basis, we thought of this topic to be both interesting and useful in providing more insight for personalization of the content according to the various users.

The dataset used in this project is provided by Airbnb and contains a list of users along with their demographics. We used 2 of the CSV files namely, train_users_2 and test users as our dataset. Unlike Kaggle, we have chosen to predict the new user's first booking destination at the granularity of city rather than the country. There are 10 destination cities to choose from, including, Dubai, Paris, Mumbai, Washington DC, Las Vegas, Shanghai, Rio De Janeiro, Chicago, Singapore, London and all the rest are labelled as "others" and users who have not made a booking are categorized with "NDF" label in the destination field. In the dataset, Airbnb provides a labelled training set of 213451 entries with 16 properties and a test set of 62097 with 15 properties to predict. Based on user's behavior and the general factors that affect the planning of a trip to a destination, we added few features to the dataset such as 'Personality', 'Types of Trips', 'Budget', 'Length of Trip' and 'Preferred Month'.

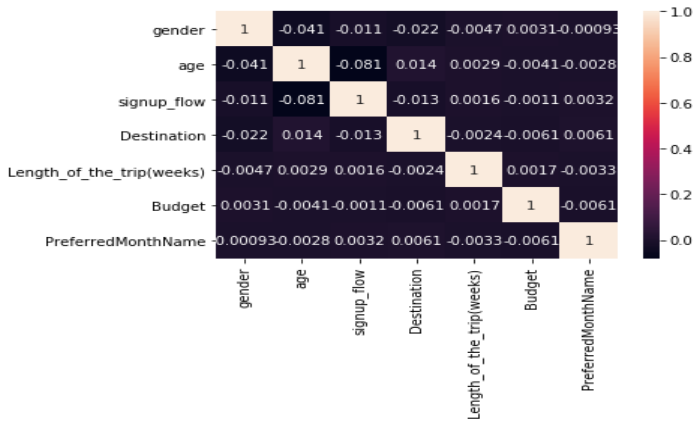
As the dataset contained number of inconsistent and null values which could have hampered the prediction, we performed data cleaning and dropped the irrelevant entries. Further Exploratory Data Analysis (EDA) was performed, resulting into statistical insights and visualizations on the trained dataset. For model selection, we started with supervised learning algorithms, and then narrowed the category down to utilize the classification methods. By modeling the data with different classifiers like Naive Bayes, Multilayer perceptron classifier, Random Forest Classifier, Support Vector Machine Algorithm and Recurrent Neural Networks; we are convinced that *RNN* seems to work the best for this prediction.

3. Code with Documentation

Since our entire dataset is filled with categorical labels, which are the destination cities of users' fist booking, each data entry is basically a pair of an input vector and a desired output value. As our prediction is also a categorical value which is discrete, we identify this problem to be a supervised learning classification problem that can be best solved using classification methods.

3.1 Exploratory Data Analysis

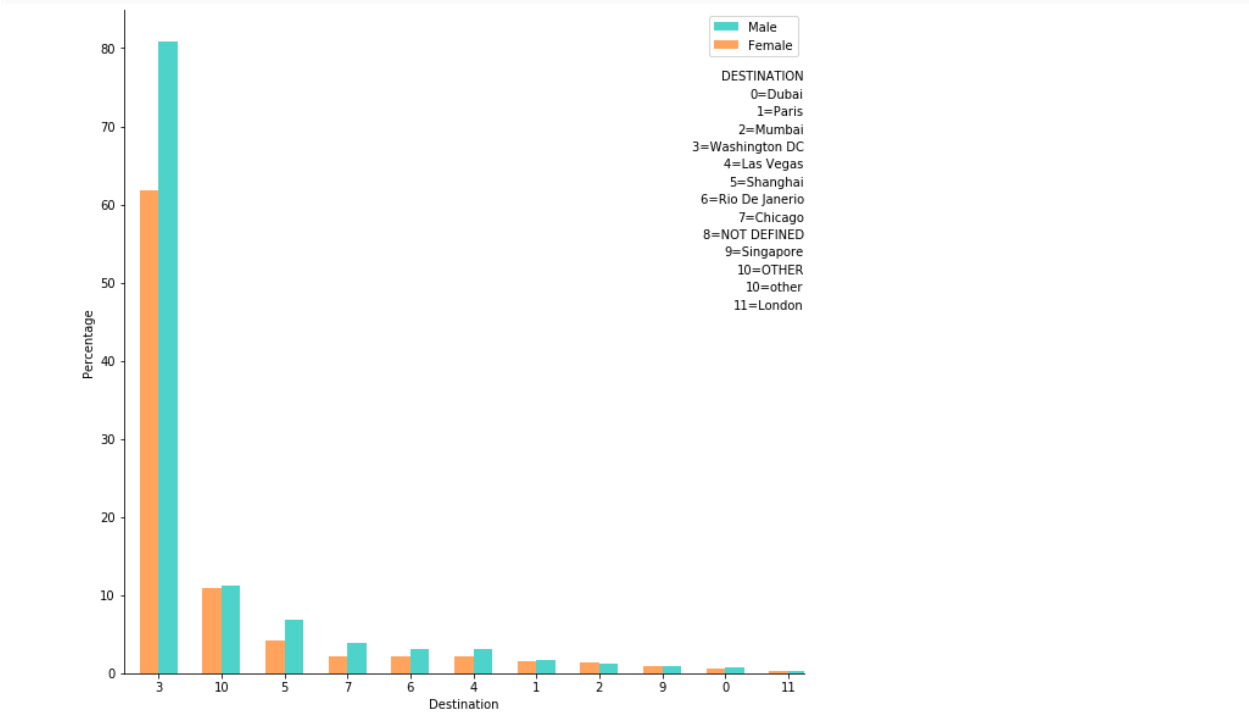
Exploratory data analysis (EDA) is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task. [12]



The parameters in the dataset are not highly correlated.

To understand the correlation between different parameters present in our dataset, we plotted a heat map and discovered that the correlation between the different parameters is not very strong.

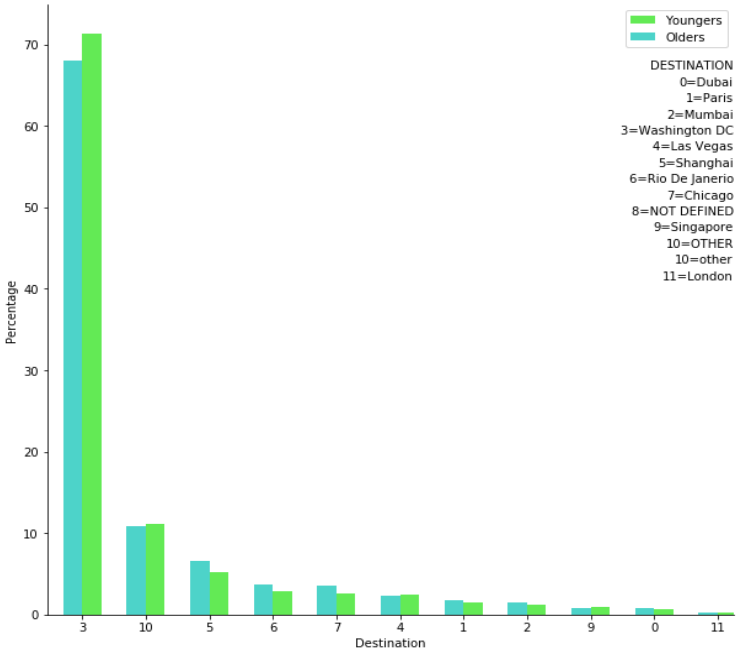
We then plotted gender of the users by splitting them in two classes ‘male’ and ‘female’ (in percentage) on X-axis against the destination cities on Y-axis using a bar chart.



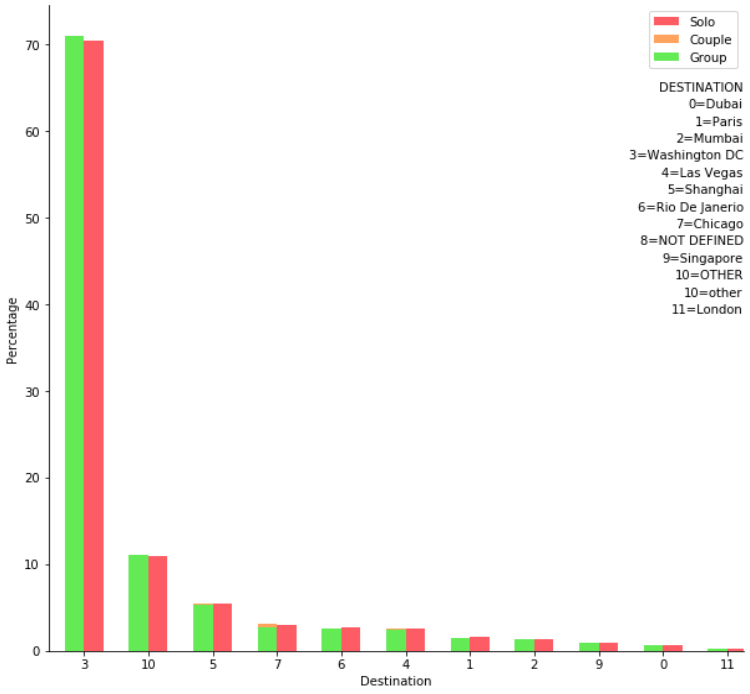
The graph portrays that about 80% of the travelers are male and maximum travelers travel to Washington DC, thus making it the most visited city. The least travelled city, according to the graph is London.

We then conducted analysis on the ‘age’ of the users mentioned and decided to split them into 2 sections: one with age above 45 as ‘Olders’ and the one with age below 45 as ‘Youngers’. This

analysis gives us an insight into what category of people want to travel to which country. The graph describes that over 70% of 'younger' and about 67% of 'older' people travel to Washington DC which again proves that Washington DC is the most visited destination city. Again, the least preferred destination by both the age categories is 'London'.



Another analysis shows that the preferred 'Type of Trip' is a 'Group' type followed by 'Solo' trips and the last being 'Couple' ones. The most preferred destination city is again Washington DC and the least preferred is 'London' again. This bar chart uses Types of Trips in percentage on Y-axis and Destination City on the X- axis.



Most of the solo and group tend to go to washington DC.

3.2 Feature Parsing

After performing EDA, we noticed that most of the features in the *train_users_2.csv* are in string format. However in order to make the model work effectively they needed to be in a numerical format. We used *'one-hot encoding'* to give some features with values between '1' and 'n' classes which is the number of types of this feature. This process could result into *overfitting* due to the increased number of features and thus we progress with feature selection as a solution to this problem. [2]

```
In [215]: print(cols)
```

```
['country_Destination', 'age', 'id', 'date_account_created', 'timestamp_first_active', 'date_first_booking', 'gender', 'signup_
method', 'signup_flow', 'language', 'affiliate_channel', 'affiliate_provider', 'first_affiliate_tracked', 'signup_app', 'first_
device_type', 'first_browser', 'Personality', 'Types_of_trips', 'Length_of_the_trip(weeks)', 'Budget', 'PreferredMonthName']
```

```
In [216]: #one hot encoding to prepare for modelling
encoding = pd.get_dummies(travel_train_modifed.iloc[:,6:17],columns =travel_train_modifed.iloc[:,6:17].columns, prefix=list(trave
onehot = pd.concat([travel_train_modifed.iloc[:,:6],encoding,travel_train_modifed.iloc[:,17:]],axis=1)
```

```
In [217]: onehot.head()
```

Out[217]:

	country_Destination	age	id	date_account_created	timestamp_first_active	date_first_booking	gender_0	gender_1	gender_2	signup_method_basic
2	3	56.0	4ft3gnwmtx	9/28/2010	2.01e+13	8/2/2010	0	1	0	1
3	10	42.0	bjjt8pjhuk	12/5/2011	2.01e+13	9/8/2012	0	1	0	0
6	3	46.0	lsw9q7uk0j	1/2/2010	2.01e+13	1/5/2010	0	1	0	1
7	3	47.0	0d01nltbrs	1/3/2010	2.01e+13	1/13/2010	0	1	0	1
8	3	50.0	a1vcnhxeij	1/4/2010	2.01e+13	7/29/2010	0	1	0	1

5 rows × 138 columns

3.3 Feature Selection

After one-hot-encoding', the number of features in our data set increased to 138 which could have contributed to over-fitting. To tackle this problem, we performed feature selection which is a process where you automatically select those features in your data that contribute most to the prediction variable or output in which you are interested. [3]

Feature importance scores can be used for feature selection in scikit-learn.[4] We used 2 ways for feature selection, namely XGBoost and Extra Trees Classifier as they automatically provide the estimates of feature importance from a trained predictive model. The reason we used both the ways is to validate the results of each of the methods and use the best of the features for prediction.

In [224]:
```
#FEATURE SELECTION - check most important features to the Extra Trees Classifier algorithm

featureImportance = pd.DataFrame(model.feature_importances_,onehot.columns[6:138],columns=['feature_importance']).sort_values(["f
featureImportance.head(10)
```

Out[224]:

	feature_importance
PreferredMonthName	0.257207
Budget	0.239658
Length_of_the_trip(weeks)	0.134466
Types_of_trips	0.095842
Personality_0	0.016658
Personality_3	0.016317
signup_method_basic	0.015477
Personality_2	0.015351
signup_method_facebook	0.014740
Personality_1	0.014711

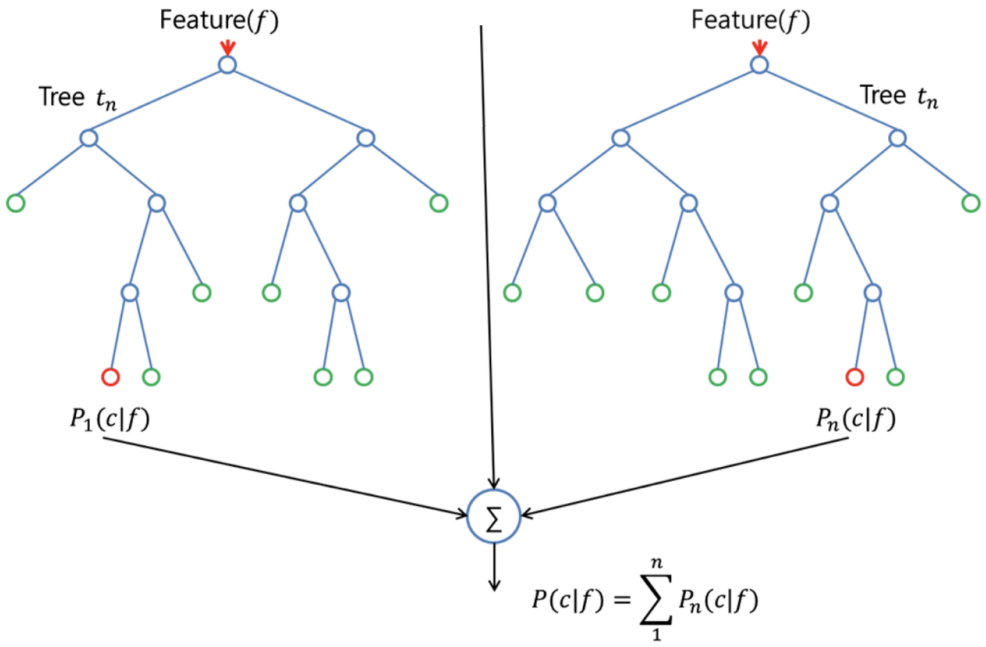
In [222]:
```
#FEATURE SELECTION - check most important features to the XGBoost algorithm
featureImportance = pd.DataFrame(model.feature_importances_,onehot.columns[6:138],columns=['feature_importance']).sort_values(["f
featureImportance.head(10)
```

Out[222]:

	feature_importance
PreferredMonthName	0.093603
Budget	0.089500
Length_of_the_trip(weeks)	0.039508
Types_of_trips	0.028719
gender_0	0.027200
gender_1	0.022641
Personality_2	0.022337
first_browser_Chrome	0.021729
signup_method_basic	0.020362
signup_flow_2	0.019754

3.4 Random Forest Classifier (RFC)

Random Forest Classifier creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object. [5]



The reasons we chose Random Forest Classifier for our dataset are:

- As most of the features in our data set are categorical values and can be handled using RFC
- RFC deals with classification tasks which align with our problem
- RFC is known to handle missing values

Before Feature Selection:

Random Forest Classifier

In [235]:
```
#train test split dataset to measure performance. Original test_users dataset provided by airbnb does not come with labels
# so we need to create our own test set
x_train,x_test,y_train,y_test = train_test_split(onehot,onehot['country_Destination'],test_size=0.25,random_state=1)
```

In [236]:
```
#Random forest classification - instantiate classifier

from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
```

In [237]:
```
#fit RF classifier

clf.fit(x_train.iloc[:,6:138],y_train)
```

Out[237]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [238]:
```
#predict and store predictions in a series

preds = clf.predict(x_test.iloc[:,6:138])
```

In [239]:
```
#add the series to our dataframe

x_test['predicted_country'] = preds
```

In [240]:
```
#import packages used for model evaluation

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score,f1_score,recall_score
def score(true,pred):
    return(precision_score(true,pred,average='weighted'),
           recall_score(true,pred,average='weighted'),
           f1_score(true,pred,average='weighted'))
```

In [241]:
```
#get overall accuracy score for the RF model

accuracy_score(x_test['country_Destination'],x_test['predicted_country'])
```

Out[241]:
```
0.6621621621621621
```

After Feature Selection:

In [244]:
```
#split the data again using only the top 100 features

x_train,x_test,y_train,y_test = train_test_split(onehot.loc[:,list(featureImportance[:5].index)],
                                                 onehot['country_Destination'],test_size=0.25,random_state=1)
```

In [245]:
```
#Generate Predictions for RF classifier with 100 features

clf = RandomForestClassifier()
clf.fit(x_train,y_train)
preds = clf.predict(x_test)
```

In [246]:
```
#New accuracy score - .3% improvement in classification

RF = accuracy_score(y_test,preds)
RF
```

Out[246]:
```
0.69923399335164038
```

Using RFC, we were able to achieve an *accuracy rate of 70%* as compared to the accuracy of *62% provided by Kaggle.*

We validated the model's accuracy using *Cross Validation*.

```
In [247]: from sklearn.model_selection import cross_val_score, cross_val_predict
          from sklearn.ensemble import RandomForestClassifier
          cross_val_score(clf, x_train, y_train)

Out[247]: array([ 0.68981214,  0.68685409,  0.68974989])
```

From the above Randomforestclassifier, we are able to achieve accuracy about 70%

3.5 Multinomial Naive Bayes Classifier

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes theorem with the naive assumption of independence between every pair of features. The classification result replies on the prior probability of classes and the posterior probability of features given labels. [8]

Multinomial Naive Bayes classifier is a specific instance of a Naive Bayes classifier which uses a multinomial distribution for each of the features. Factors that influence the performance of Naive Bayes classifier are: *even distribution of nodes in each feature vector* and *even dependence distribution of feature vectors*.

With a multinomial event model, feature vectors represent the frequencies with which certain events have been generated by a multinomial (p_1, p_2, ...p_n) where p_i is the probability that event *i* occurs. A feature vector x=(x_1, x_2, ...x_n) is then a histogram, with x_i counting the number of times event *i* was observed in a particular instance. The likelihood of observing a histogram **x** is given by [9]

$$p(\mathbf{x} \mid C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i p_{ki}{}^{x_i}$$

Our feature selection criteria guaranteed that our feature set satisfy both of the requirements. We started by splitting the dataset into *training(75%)* and *testing(25%)* and tried out different values for parameters:

a) *alpha*(Additive smoothing parameter)
b) *fit_prior* (Whether to learn class prior probabilities or not)
c) class_prior **(**Prior probabilities of the classes)

We were able to achieve a better accuracy of *81.3%* when '*alpha=1.0, class_prior=None, fit_prior=True'* was used.

Multinomial naive bayes classifier

```
In [248]: df=onehot
          df.drop(['date_account_created'], axis=1,inplace=True)
          df.drop(['date_first_booking'], axis=1,inplace=True)
          df.drop(['timestamp_first_active'], axis=1,inplace=True)
          df.drop(['id'], axis=1,inplace=True)

In [249]: x_train,x_test,y_train,y_test = train_test_split(df,df['country_Destination'],test_size=0.25,random_state=1)

In [268]: ## Multinomial Naive Bayes
          from sklearn.naive_bayes import MultinomialNB
          mnb = MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
          mnb.fit(x_train,y_train)
          preds = mnb.predict(x_test)

In [269]: MNB = accuracy_score(y_test,preds)
          MNB

Out[269]: 0.81305101893337184
```

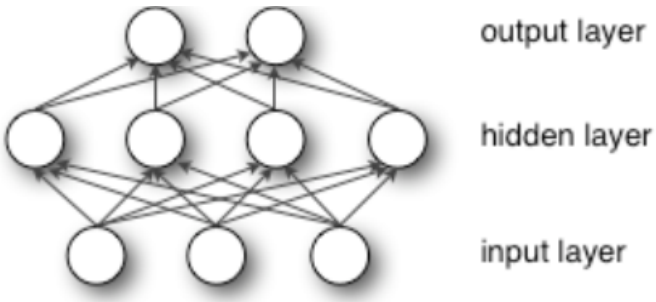
We used *10-fold cross validation* for checking the performance of model and concluded that a better performance was achieved as compared to Random Forest model(70%).

```
In [270]: from sklearn.cross_validation import cross_val_score
          mnb = MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
          scores = cross_val_score(mnb, x_train, y_train, cv=10, scoring='accuracy') #cv is cross validation
          print(scores)

          [ 0.81227437  0.81198844  0.81218396  0.81290633  0.8099711   0.81137075
            0.81180723  0.81296698  0.81243973  0.81066088]
```

3.6 Multi-layer Perceptron classifier (MLP)

An MLP (or Artificial Neural Network - ANN) with a single hidden layer can be represented graphically as follows:



[10]

The supervised learning problem of the MLP can be solved with the *back-propagation algorithm*. The algorithm consists of two steps.

- In the *forward pass*, the predicted outputs corresponding to the given inputs are evaluated.

- In the *backward pass*, partial derivatives of the cost function with respect to the different parameters are propagated back through the network.

The chain rule of differentiation gives very similar computational rules for the backward pass as the ones in the forward pass. The network weights can be adapted using any gradient-based optimization algorithm. The whole process is iterated until the weights have converged. [11] MLP Classifier trains iteratively since at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters.

As with every model, we started by splitting into train and test datasets and experimented the MLP Classifier with different values for parameters like:

a. hidden_layer_sizes (The i^{th} element represents the number of neurons in the i^{th} hidden layer)
b. max_iter (Maximum number of iterations. The solver iterates until convergence)
c. alpha (L2 penalty (regularization term) parameter)
d. solver (The solver for weight optimization)
e. verbose (Whether to print progress messages to stdout)
f. random_state (If int, random_state is the seed used by the random number generator)
g. tol (Tolerance for the optimization)

Better results are obtained with parameter values like: *hidden_layer_sizes=(100,100,100) with 3 hidden layers, max_iter=500, alpha=0.0001, solver='sgd', verbose=10, random_state = 21, tol=0.000000001*.

The accuracy achieved using this model is 88.6% which is better than results achieved using *multinomial naive bayes(81.3%)*

In [72]:
```
x_train,x_test,y_train,y_test = train_test_split(df,df['Destination'],test_size=0.25,random_state=1)
```

In [73]:
```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(100,100,100), max_iter=500, alpha=0.0001,
                    solver='sgd', verbose=10,  random_state=21,tol=0.000000001)
mlp.fit(x_train,y_train)
preds = mlp.predict(x_test)
```

```
Iteration 1, loss = 1.00003994
Iteration 2, loss = 0.73196776
Iteration 3, loss = 0.61820093
Iteration 4, loss = 0.50979331
Iteration 5, loss = 0.41594654
Iteration 6, loss = 0.35111394
Iteration 7, loss = 0.30833257
Iteration 8, loss = 0.27549111
Iteration 9, loss = 0.24974298
Iteration 10, loss = 0.22678800
Iteration 11, loss = 0.20556482
Iteration 12, loss = 0.18803684
Iteration 13, loss = 0.17220476
Iteration 14, loss = 0.15737153
Iteration 15, loss = 0.14654126
Iteration 16, loss = 0.13355967
Iteration 17, loss = 0.12404532
Iteration 18, loss = 0.11447166
Iteration 19, loss = 0.10682085
Iteration 20, loss = 0.09878233
Iteration 21, loss = 0.09273527
Iteration 22, loss = 0.08753530
Iteration 23, loss = 0.08132759
Iteration 24, loss = 0.07689155
Iteration 25, loss = 0.07306404
Iteration 26, loss = 0.06871595
Iteration 27, loss = 0.06585007
Iteration 28, loss = 0.27113268
Iteration 29, loss = 0.43991060
Iteration 30, loss = 0.20771834
Training loss did not improve more than tol=0.000000 for two consecutive epochs. Stopping.
```

In [74]:
```
multiLayerPerceptron = accuracy_score(y_test,preds)
multiLayerPerceptron
```

Out[74]:
```
0.9457291516115045
```

Again, we cross-validated the accuracy of this model to confirm its working.

In [75]:
```
from sklearn.cross_validation import cross_val_score
mlp = MLPClassifier(hidden_layer_sizes=(100,100,100), max_iter=500, alpha=0.0001,
                    solver='sgd', verbose=10,  random_state=21,tol=0.000000001)
scores = cross_val_score(mlp, x_train, y_train, cv=10, scoring='accuracy') #cv is cross validation
print(scores)
print(scores.mean())
```

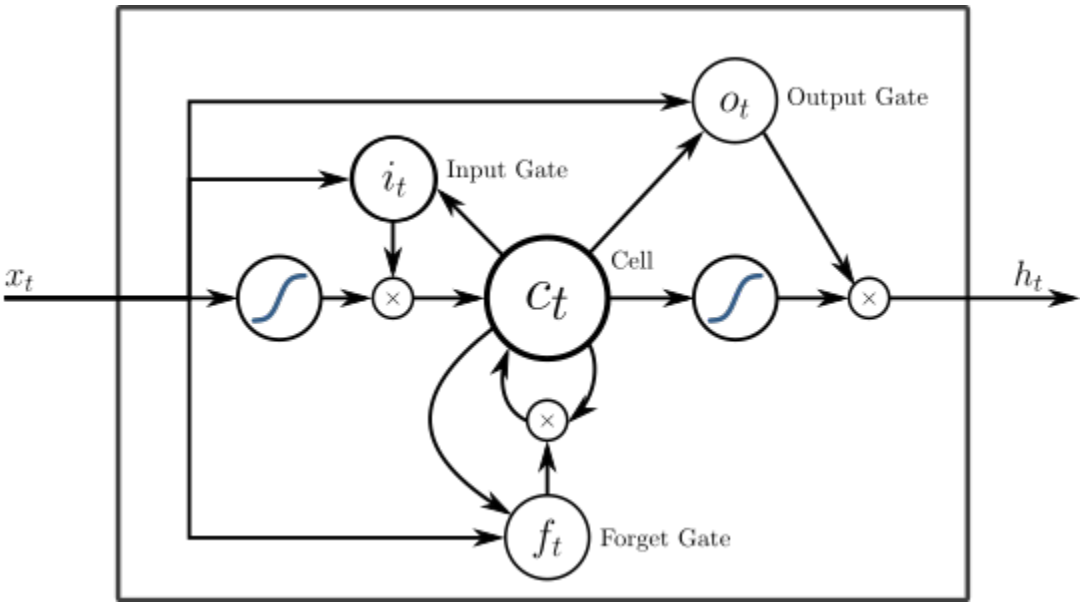
```
Iteration 15, loss = 0.16704767
Iteration 16, loss = 0.15454737
Iteration 17, loss = 0.14376543
Iteration 18, loss = 0.13387223
Iteration 19, loss = 0.12436089
Iteration 20, loss = 0.11524670
Iteration 21, loss = 0.10711880
Iteration 22, loss = 0.10098867
Iteration 23, loss = 0.09450681
Iteration 24, loss = 0.08909310
Iteration 25, loss = 0.08442529
Iteration 26, loss = 0.07960216
Iteration 27, loss = 0.72114618
Iteration 28, loss = 0.55288650
Iteration 29, loss = 0.44032136
Training loss did not improve more than tol=0.000000 for two consecutive epochs. Stopping.
[ 0.88062575  0.95859413  0.88080905  0.94317361  0.87861272  0.83883402
  0.87373494  0.87177633  0.86547734  0.87819585]
0.886983373535
```

The accuracy achieved using this model is 94.5% which is better than results achieved using multinomial naive bayes(81.3%)

3.7 Recurrent Neural Network (RNN)

Recurrent neural networks (RNNs) are connectionist models with the ability to selectively pass information across sequence steps, while processing sequential data one element at a time. Thus, they can model input and/or output consisting of sequences of elements that are not independent. [7]

As every row of our dataset together made complete sense as a sentence, we used Recurrent Neural Network on it. This is because it would provide more accuracy, since we can include information about the sequence of words (*every word will be a feature (column) of the row*). We used *Long Short-Term Memory (LSTM*), a specific recurrent neural network (RNN) architecture that was designed to model temporal sequences and their long-range dependencies more accurately than conventional RNNs. The architecture of LSTM cell is composed of a memory cell, an input gate, an output gate and a forget gate as shown below. [13]



Here, we passed words to an embedding layer (we need an embedding layer because we have tens of thousands of words, so we'll need a more efficient representation for our input data than one-hot encoded vectors) as it is good to have an embedding layer and let the network learn the embedding table on it's own. From the embedding layer, the new representations will be passed to LSTM cells. These will add recurrent connections to the network so we can include information about the sequence of words in the data. The output layer will just be a single unit then, with a *ReLU* activation function.

A Graph is built using the hyperparameters such as *lstm_size*(Number of units in the hidden layers in the LSTM cells), *lstm_layers*(Number of LSTM layers in the network), *batch_size*(The number of feature to feed the network in one training pass), *learning_rate* with 256, 1, 100, 0.001 as values respectively. For the network, we have passed our feature vectors and each batch will be batch_size vectors.

To create a basic LSTM cell for the graph, *tf.contrib.rnn.BasicLSTMCell* was used and *dropout* was added to the cell with *tf.contrib.rnn.DropoutWrapper* to wrap the cell in another cell. In order for network to have better performance, we added more layers to create multiple layers of LSTM cells with *tf.contrib.rnn.MultiRNNCell.*

In order to run the data through the RNN nodes, RNN cell (multiple layered LSTM cell) is passed through *tf.nn.dynamic_rnn* along with the inputs to the network.[14]

We created an initial state to pass to the RNN and this is the cell state that is passed between the hidden layers in successive time steps. *tf.nn.dynamic_rnn* is also used to add the forward pass through the RNN and vectors are passing in from the embedding layer. We played with different hyperparameters such as *activation function, cost function, Gradient Estimation, Epochs, Network Architecture, network initialization* and concluded that using *Rectified linear unit (ReLU)* as Activation function, *Quadratic cost (mean-square error)* as Cost function, *AdamOptimizer* as Gradient Estimation, *Epochs* = 5.
With these parameters we are able to achieve accuracy of *99.7%* using RNN.

Activation function - Rectified linear unit (ReLU)
Cost function - Quadratic cost (mean-square error)
Gradient Estimation - GradientDescentOptimizer

In [96]:
```
with graph.as_default():
    predictions = tf.contrib.layers.fully_connected(outputs[:, -1], 1, activation_fn=tf.nn.relu)
    cost = tf.losses.mean_squared_error(labels_, predictions)

#   optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
    optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

In [100]:
```
test_acc = []
with tf.Session(graph=graph) as sess:
    saver.restore(sess, tf.train.latest_checkpoint('checkpoints'))
    test_state = sess.run(cell.zero_state(batch_len, tf.float32))
    for ii, (x, y) in enumerate(get_batches(test_x, test_y, batch_len), 1):
        feed = {inputs_: x,
                labels_: y[:, None],
                keep_prob: 1,
                initial_state: test_state}
        batch_acc, test_state = sess.run([accuracy, final_state], feed_dict=feed)
        test_acc.append(batch_acc)
    print("Test accuracy: {:.3f}".format(np.mean(test_acc)))
```

```
INFO:tensorflow:Restoring parameters from checkpoints\destination_preds.ckpt
Test accuracy: 0.720
```

3.8 Support Vector Machine (SVM)

In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyperplane that differentiate the two classes very well. [6]

In SVM, training involves the minimization of the error function:

$$\frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i$$

subject to the constraints:

$$y_i \left(w^T \phi(x_i) + b \right) \geq 1 - \xi_i \text{ and } \xi_i \geq 0, i = 1, \ldots, N$$

where C is the capacity constant, w is the vector of coefficients, b is a constant, and ξ_i represents parameters for handling non- separable data (inputs). The index i labels the N training cases. Note that $y \in \pm 1$ represents the class labels and xi represents the independent variables. The kernel ϕ is used to transform data from the input (independent) to the feature space. It should be noted that the larger the C, the more the error is penalized. Thus, C should be chosen with care to avoid overfitting [13].

The dataset is split into *train (75%)* and *test dataset (25%)* to measure performance. Initially we trained the data and were able to achieve an accuracy of 99.1% using default hyperparameters: *(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None).*

When different parameters are tried by changing *C value, kernel, degree* and *gamma,* accuracy of 96.8% was achieved using *kernel = 'rbf', C = 3, gamma = 0.1.* 98.7% accuracy is achieved while using *kernel='poly' with SVM.*
To validate our model, we used *10-fold cross validation* and checked its performance.

Training and fitting SVM

In [102]:
```
from sklearn.svm import SVC
from sklearn import metrics
svc=SVC(kernel='rbf', C=3, gamma=0.1)
svc.fit(x_train,y_train)
y_pred=svc.predict(x_test)
print('Accuracy Score:')
print(metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy Score:
0.949631449631
```

In [103]:
```
svc=SVC(C=1.0, kernel='rbf', degree=3, gamma='auto')
svc.fit(x_train,y_train)
y_pred=svc.predict(x_test)
print('Accuracy Score:')
print(metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy Score:
0.987642722937
```

Performing K-fold cross validation with different kernels

CV on rbf kernel

In [104]:
```
from sklearn.cross_validation import cross_val_score
svc=SVC(kernel='rbf')
scores = cross_val_score(svc, x_train, y_train, cv=10, scoring='accuracy') #cv is cross validation
print(scores)
```

```
[ 0.98796631  0.98892634  0.98362629  0.98458945  0.98747592  0.98385931
  0.98626506  0.98602073  0.98601736  0.98769899]
```

4. Results

Since this is a classification problem, we worked on our dataset with multiple classifier models and their hyperparameters. The accuracy from each of the methods along with the hyperparameters used is mentioned in the table below

#	Method	Hyperparameters	Accuracy
1	Random forest	n_estimators=10, criterion='gini', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, min_impurity_decrease=0.0, bootstrap=True, n_jobs=1	70%
2	Multinomial naive bayes classifier	alpha=1.0, class_prior=None, fit_prior=True	81.30%
3	Multi-layer Perceptron classifier (MLP)	hidden_layer_sizes=(100,100,100), max_iter=500, alpha=0.0001, solver='sgd', verbose=10, random_state=21,tol=0.000000001	94.50%
4	RNN	epochs = 5, Activation function =ReLU, Cost function=Quadratic cost (mean-square error), Gradient Estimation =AdamOptimizer **Network Architecture** lstm_size = 256, lstm_layers = 1, batch_len = 100, learning_rate = 0.001	99.7%
5	SVM	kernel='rbf', C=3, gamma=0.1	95%
		C=1.0, kernel='rbf', degree=3, gamma='auto'	98.70%

With the above working models and the comprehensive analysis performed on the dataset, we conclude that *RNN provides us with the best accuracy results* and hence provides an apt solution to our classification problem.

5. Discussion

From EDA, we were able to conclude that Washington DC is the most visited destination among the users with respect to age and gender of the users. The features that are most helpful to this prediction task are PreferredMonthName, Budget, Length_of_the_trip(weeks), gender, Types_of_trips, Personality, first_browser, sign_up and first device time.

In this paper, we implemented five different supervised classifier models to provide solution for our classification problem of predicting new user's destination booking. We experimented with various hyperparameters of all the models to achieve a better accuracy score. As compared to Kaggle's best accuracy score of 66%, we have achieved accuracy score of 99.7% with Recurrent Neural network.

In the future, we can further enhance this project by reducing its prediction granularity to specific famous locations in the city and also to design a strategy to deal with the imbalanced classes by determining whether some NDFs in the training dataset are hosts on AirBnB on the basis of their web activity.

6. Citations

[1] https://en.wikipedia.org/wiki/Airbnb

[2] https://cseweb.ucsd.edu/~jmcauley/cse255/reports/fa15/045.pdf

[3] https://machinelearningmastery.com/feature-selection-in-python-with-scikit-learn/

[4] https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/

[5] https://medium.com/machine-learning-101/chapter-5-random-forest-classifier-56dc7425c3e1

[6] https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

[7] https://arxiv.org/pdf/1506.00019.pdf

[8] https://cseweb.ucsd.edu/~jmcauley/cse255/reports/fa15/045.pdf

[9] https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[10] http://deeplearning.net/tutorial/mlp.html

[11] http://www.helsinki.fi/~ahonkela/dippa/node41.html

[12] https://en.wikipedia.org/wiki/Exploratory_data_analysis

[13] http://www.statsoft.com/Textbook/Support-Vector-Machines

[14] https://www.kaggle.com/lusob04/titanic-rnn

[15] Stuart J. Russell and Peter Norvig. 2003. Artificial Intelligence: A Modern Approach (2 ed.). Pearson Education. *See p. 499 for reference to "idiot Bayes" as well as the general definition of the Naive Bayes model and its independence assumptions*

[16]http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

[17]http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html