Machine Learning - Prediction and Recommendation

(excerpt from project report)

a) Data:

- In our prediction and recommendation system, we used the dataset of a Kaggle competition (The link of this is shared in the references section). The provided data had lots of tables. However, in this project we have worked only on the most important two features and hence we focussed only on the tables related to those features, which were:
  - Users.csv
  - Events.csv
  - Training.csv
  - test.csv
- users.csv consisted of the user information like location etc, events.csv consisted of events' information like location etc. Training.csv and test.csv had same columns like userId, eventId, interested etc.
- Towards the end of our project we created our own training.csv with two engineered features in order to train our model. Feature-engineering will be explained in the following sections.

b) Data preprocessing:

- The dataset required preprocessing as there were some inconsistent format and many missing. We handled the preprocessing in the following way:

- Missing values:
  We replaced the missing values with NA in the users data. In case of training data, the "interested" column (the column that served as the groundtruth) had also missing values. We removed the rows containing missing "interested" values, since we figured that they won't contribute to the "learning" part of the prediction-recommendation system.

- Inconsistent location format:
  The users and events tables had inconsistent location format. We figured that for measuring distance between the user's and the event's location, only the city and the state is enough. Almost all of the rows in both the tables had this information so the inconsistent location no more posed a challenge for us.

- For all the data preprocessing, we used IBM Watson Studio by creating a Data Refinery project (in the Data Engineering tab). It made all the preprocessing operations very easy to perform.

c) Feature Engineering:

- Based on the data obtained from the Kaggle competition, there was scope for lot of features, but we narrowed in to just two features we thought were most important, i.e. proximity and relevance, for a pair of user and event (u,e). Each of the feature and the method of engineering it is as follows:

- Proximity:
    - This feature is basically the distance between the user's location and the event's location. We figured that a user is likely to click on "interested" if the event is within the user's proximity.
    - To generate this feature for each pair of user and event, we have used the geopy library of python. By providing the city and state of every user and event to this library's functions, we get the distance in miles for every pair.
    - In the application, when the user selects either prediction/ recommendation option, user's location(which is taken from the user's profile) and event location(which is mentioned in the event's details) are provided to the python code. The python code generates the proximity feature, along with the relevance feature (described later in the report) and accesses the trained model exposed as an API(details are mentioned later in the report) in order to generate the result.

- Relevance:
    - This feature is basically the cosine similarity between two events. In case of prediction system, when a user enters the details of the events and clicks on "predict" button, the event's description is compared with that of other past events in Evento's database and based on how many users clicked "interested" for those past events, a number is predicted for this event.
    - In case of recommendation system, when the user clicks on "view recommendations", every upcoming event is compared with the past events of the user (for whom we are recommending), for which the user clicked on "interested". Only those events are displayed as result for which the cosine similarity with the past interested events is higher.

    - For getting the cosine similarity between two events, we have used TFIDF vectorizer of scikit-learn library. The TfidfVectorizer tokenizes the event's description, learns the vocabulary(both using "fit"operation) and inverse description frequency weightings, and we are able to encode descriptions of upcoming events(using "transform" operation). This results into a sparse array for each event description. These sparse arrays are then used to find cosine similarity between the descriptions of the two events. We have used sklearn for cosine similarity.

d) Training:

- We have used IBM Watson Studio to train our model using logistic regression analysis.
- We used the following services :
    - Watson™ Studio
    - IBM® Analytics for Apache Spark for IBM Cloud
    - IBM Cloud Object Storage
    - IBM Watson™ Machine Learning
    - IBM® Db2® Warehouse on Cloud

- Following are the relevant steps from the training step using Watson Studio:
    - Create new project in Watson Studio
    - Add data assets. In this step we added our training.csv
    - Select technique for machine learning. We have selected Logistic Regression

e) Deployment

- We used IBM Watson Studio for deployment of the trained model as well. After deploying it, we were able to access our model into our web application through the API provided by Watson studio.
- To access the API we noted the service credentials(url, username and password) provided by Watson. These service credentials were then used in a Python code to get required output from the trained model's API.