Machine learning for Facebook Check Ins Prediction

COMP9417 Machine Learning Project

INTRODUCTION

With the development of Internet, social networks are playing a more and more important role in our daily life. Facebook, known as one of the fastest growing social networks worldwide, launched a machine learning engineering competition Predicting Check Ins with Kaggle. The aim for this competition is to identify the correct place for check ins which helps the company to know where customers are more likely to visit and these information gives them some suggestion on advertising and promotion.

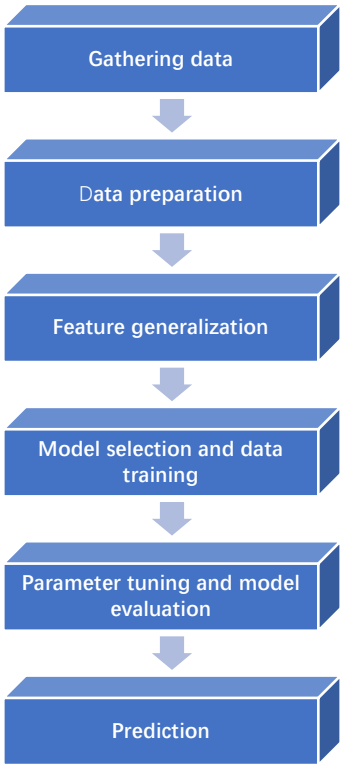
The link of this competition is https://www.kaggle.com/c/facebook-v-predicting-check-ins.

In this competition, Facebook created an artificial world composed of more than 100,000 locations in a 10km by 10km square. For a given set of coordinates, your task is to return a list of rankings of the most likely locations. The data is made to resemble a location signal from a mobile device, which give you a chance to experience handle real data that includes some inaccurate and noisy values. Inconsistent and incorrect location data can disrupt the experience of services such as Facebook Check In.

IMPLEMENTATION

In this competition, we are going to predict which business a user is checking into based on their location, accuracy, and timestamp. The train and test dataset are split based on time, and the public/private leaderboard in the test data are split randomly. The source data train.csv and test.csv has been provided with six column. The row_id which gives the id of the check-in event. X and Y are two coordinates describing the location. Meanwhile, accuracy and time indicate the accuracy of the location and when the user check in. The target is place_id which shows the id of the business that we are predicting. There is no concept of a person in this dataset. All the row_id's are events, not people. Some of the columns, such as time and accuracy, are intentionally left vague in their definitions which are considered as part of the challenge.

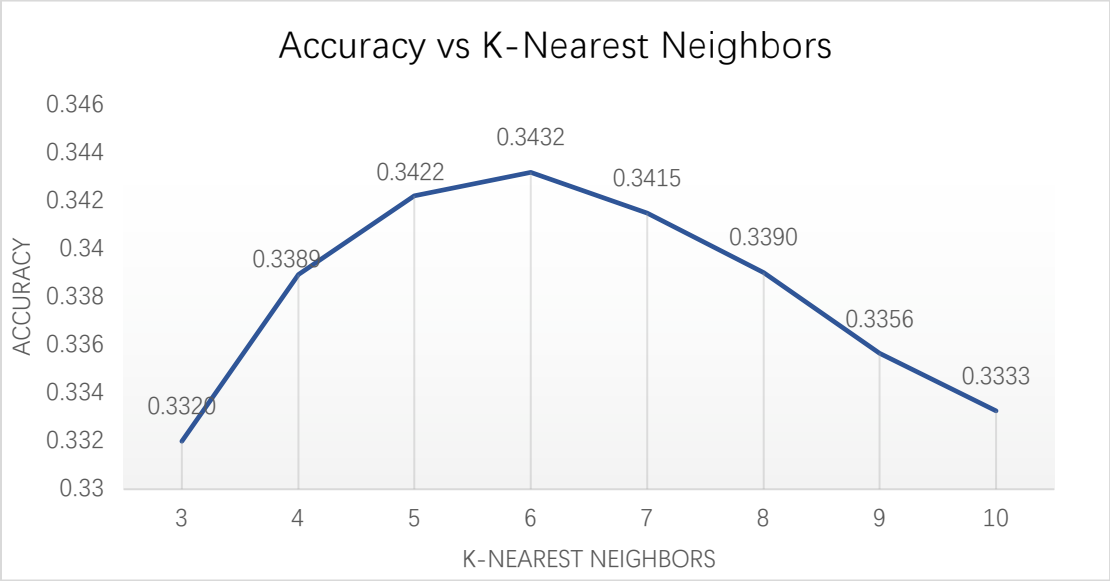
The step of machine learning for our task is described as follow:



The first algorithm we use is KNN. K-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. This will be very helpful to deal with these real world dataset that do not follow mathematical theoretical assumptions. First, store all training examples xi, f(xi). Given query instance xq, first locate nearest training example xn, then estimate $\hat{f}(x_q) \leftarrow f(x_n)$. For given xq, take vote among its k nearest neighbours (if discrete-valued target function) .Then, take mean of f values of k nearest neighbours (if real-valued) $\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$ .For each training example ⟨xi,f(xi)⟩ add the example to the list training examples. Classification algorithm Given a query instance xq to be classified, Let x1 . . . xk be the k instances from training examples that are nearest to xq by the distance function ， Return $\hat{f}(x_q) \leftarrow \arg\max_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))$.

First, we use pandas to load the date train.csv. In order to deal with the data in an easier and faster way, we shrink the range of data by splitting it into 100 sections with the same size and

randomly choose 5 percent of data from each section. By transforming the timestamp into the form of day, weekday, hour and etc. which make more sense. In the meantime, we filter out these place_ids which are rarely visited (less than 3 times) since these locations have little impact on the final result. So the feature sets are x, y, accuracy, day, weekday and hour and the target set would be place_id. Then we split the dataset and use standardscalar to preprocess the train and test dataset. The KNeighborsClassifier module help us create a KNN classifier object by passing the number of neighbors into the function. After training the model using the training set and predicting the response for test dataset, we analyze that we can achieve highest accuracy when K equals to six by comparing the actual test set values and predicted values.



We choose decision tree as our second method to handle the data. A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning.

The decision tree algorithm use Attribute Selection Measures(information gain, gain ratio, gini index) to select the best attribute to separate data. Make this attribute as a node and the separated datasets as branches. Recursively repeat these processes to build the tree until one of the following condition will match: (1) All the tuples belong to the same attribute value. (2)There are no more remaining attributes.(3)There are no more instances. For attribute

selection measures, the Attribute Selection Measures includes Information Gain, Gain Ratio, and Gini Index. These measures are the splitting criterion that partition data into the best possible manner. Following we will introduce the calculation manner of these three measures.

1.Information Gain:

Pi is the probability that an arbitrary tuple in D belongs to class Ci. Info(D) is the average amount of information needed to identify the class label of a tuple in D. |Dj|/|D| acts as the weight of the jth partition.InfoA(D) is the expected informa-tion required to classify a tuple from D based on the partitioning by A.

$$\text{Info}(D) = -\sum_{i=1}^{m} pi \log_2 pi$$

$$\text{Info}_A(D) = \sum_{j=1}^{V} \frac{|Dj|}{|D|} \times \text{Info}(D_j)$$

$$\text{Gain}(A) = \text{Info}(D) - \text{Info}_A(D)$$

The attribute A with the highest information gain, Gain(A), is chosen as the splitting attribute at node N().

2. Gain Ratio

|Dj|/|D| acts as the weight of the jth partition. V is the number of discrete values in attribute A.

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

The attribute with the highest gain ratio is chosen as the splitting attribute.

3. Gini index

pi is the probability that a tuple in D belongs to class Ci.

$$\text{Gini}(D) = 1 - \sum_{i=1}^{m} \text{Pi}^2$$

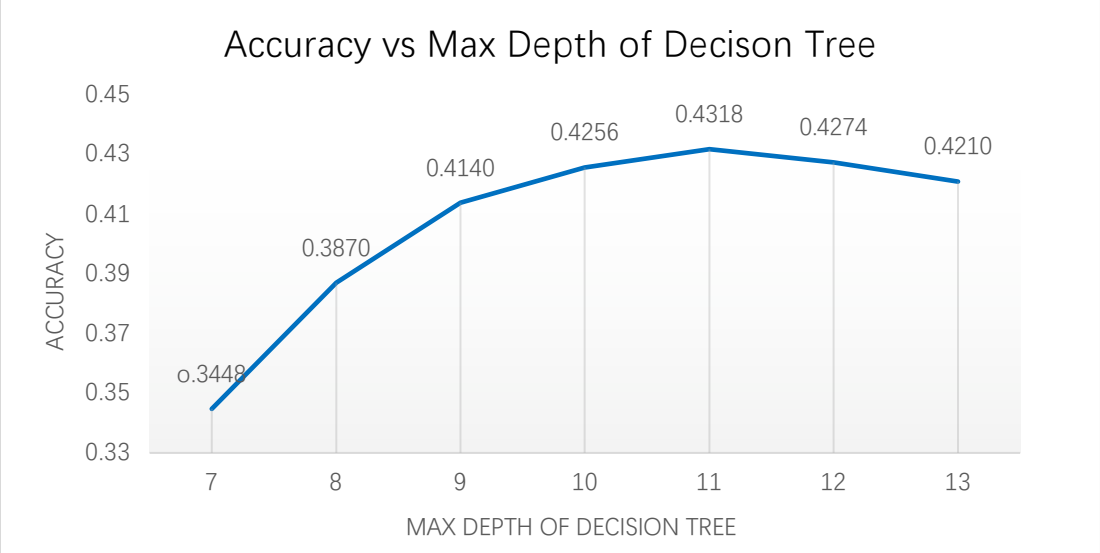
If a binary split on attribute A partitions data D into D1 and D2, the Gini index of D is:

$$\text{Gini}_A(D)=\frac{|D1|}{|D|}\text{Gini}(D_1)+\frac{|D2|}{|D|}\text{Gini}(D_2)$$

$$\Delta Gini(A) = Gini(D) - Gini_A(D).$$

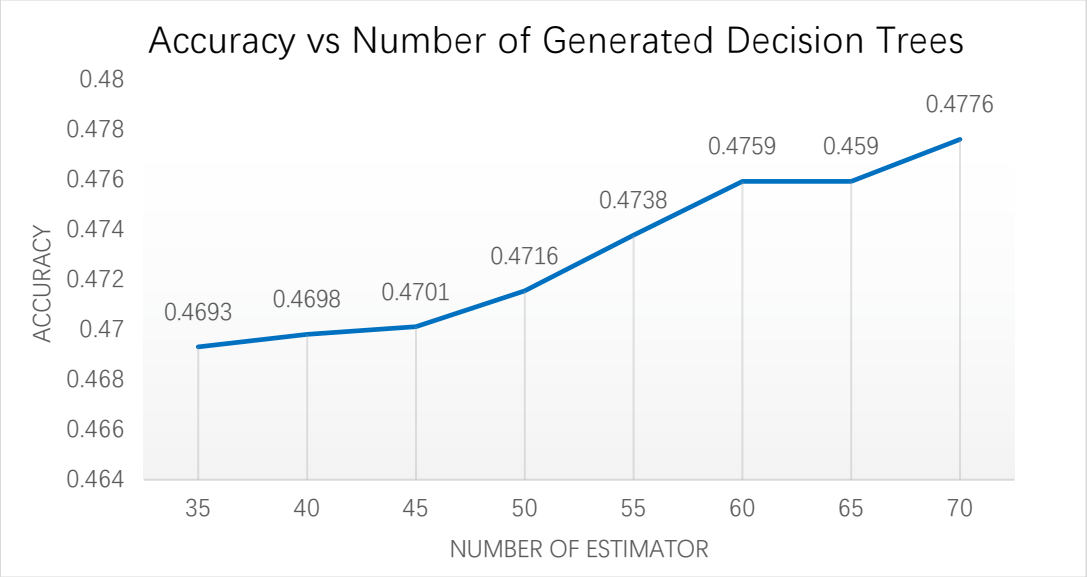
The attribute with minimum Gini index is chosen as the splitting attribute.

For decision tree optimization, there are three arguments: Criterion, which means that supported criteria are “gini” for the Gini index and “entropy” for the information gain. Max_depth is int or None, optional (default=None) or Maximum Depth of a Tree. Splitter means optional (default=”best”) or Split Strategy and we use the default in this task. According to the result, we can have the best outcome when implemented the max depth of eleven.



The third algorithm we implement is Random forest. Random forest is comprised of decision trees which is created on randomly selected data samples. Radom forest gets prediction from each tree and selects the best solution by means of voting. First of all, we select random samples from a given dataset. Secondly, construct a decision tree for each sample and get a prediction result from each decision tree. Then, perform a vote for each predicted result. Finally, select the prediction result with the most votes as the final prediction. For optimization, in sklearn library, the random forest classifier also provides some parameters such as “criterion” and “max_depth”, like the decision tree classifier. Moreover, “n_estimators” is provided to input an int to decide how many decision trees to generate for this prediction. The accuracy of prediction may increase steadily with the number of generated decision trees. However, due to a large amount of decision trees are produced in this process, the generation of the random

forest is slow.

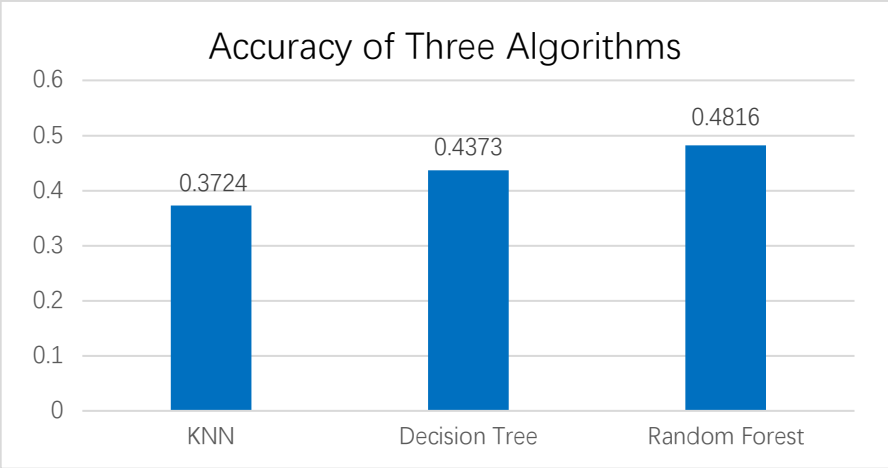


CONCLUSION

In contrast to other two algorithms, the training phase of KNN classification is much faster since it does not need to train the model for generalization. It is simple and useful when dealing with nonlinear data such as the check in prediction. But on the downside, it requires large memory for storing the entire training dataset for our prediction.

For our task, decision tree is a good choice to capture non-linear patterns. Also, it is easy to visualize and do not need to do extra data processing such as normalization. Compared to KNN, it has better accuracy and is more suitable for big datasets while noisy and imbalanced dataset can make the decision tree biased or overfitting easily.

According to the chart shown as below, in comparison to the decision tree, random forest is more accurate and robust because it is a combination of a number of decision trees although the generation of forest tree is slow as a number of decision trees are produced in this process. The prediction result is the mean of voting from these decision trees, so it avoid overfitting problem. The random forest can also use median values to replace continuous variables and compute the proximity-weighted average of missing values. This can help handle missing values.



FUTURE WORK

As you can see from our data set, there are a huge number of place ids. The number is more than 2.911802e+07 rows. This means that we need to cut a small dataset from the original dataset. Also, any algorithm which trains using a one vs all approach won't work on this dataset (unless of course you're willing to train 100k models). Thus we only combine KNN, decision tree and random forest in this task due to the large scale of the dataset. In the future work, we expect to introduce SVD, RNN and other supervised learning methods to make the result more accurate.

REFERENCES

Mohri, Mehryar; Rostamizadeh, Afshin; Talwalkar, Ameet (2012). Foundations of Machine Learning. The MIT Press. ISBN 9780262018258.

Altman, N. S. (1992). "An introduction to kernel and nearest-neighbor nonparametric regression". The American Statistician. 46 (3): 175–185. doi:10.1080/00031305.1992.10475879. hdl:1813/31637.

Rokach, Lior; Maimon, O. (2008). Data mining with decision trees: theory and applications. World Scientific Pub Co Inc. ISBN 978-9812771711.

Breiman L (2001). "Random Forests". Machine Learning. 45 (1): 5–32. doi:10.1023/A:1010933404324.

DATASET LINK

https://drive.google.com/drive/folders/1OpcWSP2ntqHM9Qm9pE6dGRcT-CJgIoHy?usp=sharing