TripType Classifier on Purchase Information

Lanxin(Lancy) Xu

Abstract:

Based on purchase information from Walmart, a classification model is built using Keras to predict TripType(the type of a shopping trip used by Walmart). This process involved necessary data cleansing, model experiments and model tuning. The best model has a multi-class accuracy of 67%.

Contents

Objective

As said in Deliverable 1, my goal is to predict the TripType based on purchase information. The original train.csv has 38 different TripType. I have tried to predict 38 classes in my model, but it seems that the large number of classes in response variable has a great influence on accuracy rate. Besides, considering the large size of my data (650k rows), it will be a better choice to predict less classes. Here, I choose to predict TripType '25', '32', '40' and 'others'. This will be explained in Data Section.

Data

1. Data Description

Train.csv: 647,055 rows and 7 columns

Fields:

- TripType - a categorical id representing the type of shopping trip the customer made. This is the ground truth that you are predicting. TripType_999 is an "other" category
- VisitNumber - an id corresponding to a single trip by a single customer
- Weekday - the weekday of the trip
- Upc - the UPC number of the product purchased
- ScanCount - the number of the given item that was purchased. A negative value indicates a product return
- DepartmentDescription - a high-level description of the item's department
- FinelineNumber - a more refined category for each of the products, created by Walmart

Note that Walmart also provides a test.csv file which lacks the TripType for competition. Therefore, it is not used here.

Field	Type	NumOfClasses	Action
TripType	Category(int)	38	Response
VisitNumber	id	-	Discard
Weekday	Category(string)	7	Feature
Upc	Category(int)	93,920	Discard
ScanCount	Numeric	-	Feature
DepartmentDescription	Category(string)	68	Feature
FinelineNumber	Category(int)	5195	Feature

Form 1.1-Field Description

2. Data Issue

The data suffers from several problems which need to be solve before moving on to model building step.

a) Multi-class Response Variable ‘TripType’

As mentioned before, the TripType field has 38 unique classes which, if transformed into dummy variables, will result in 38 columns as response variable. Such a large number of classes in y will have a negative effect on prediction accuracy. Therefore, I need to reduce it.

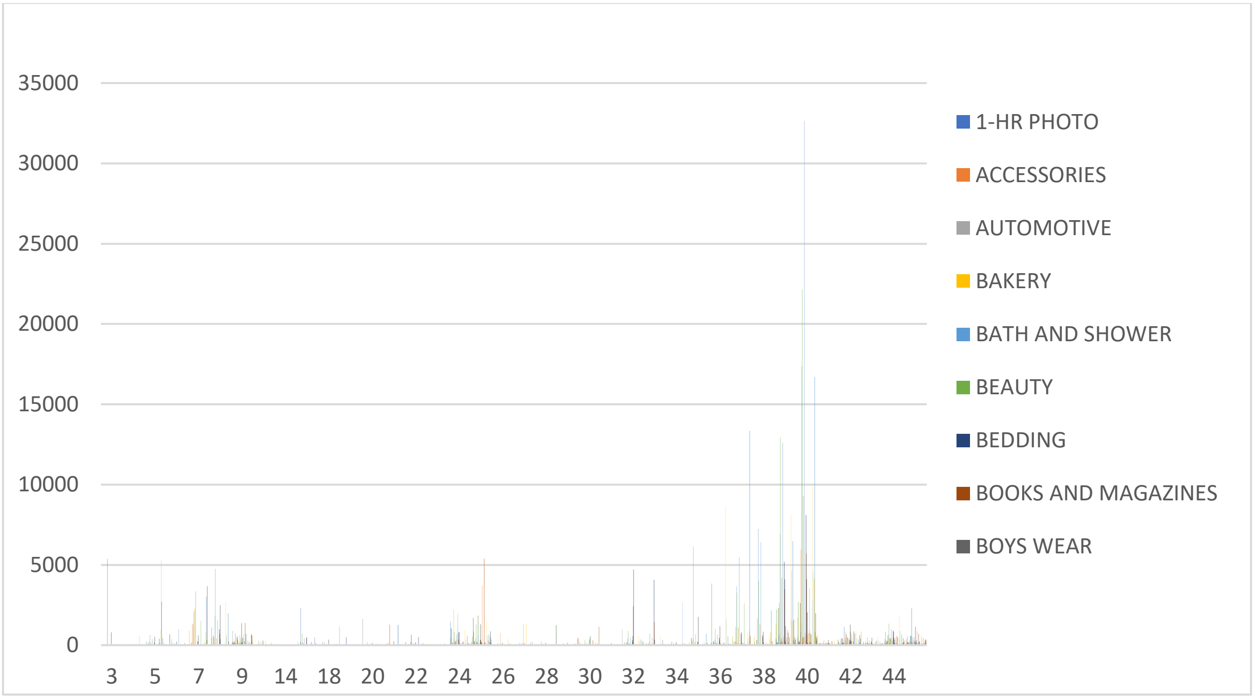
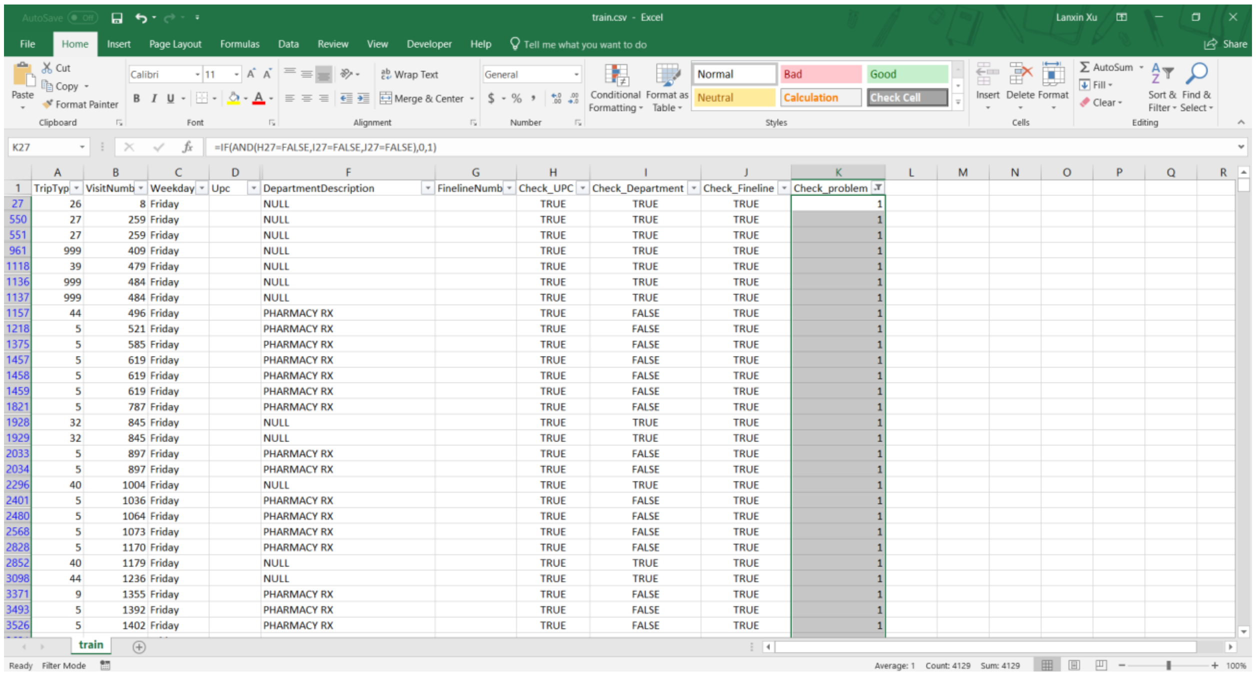


Figure 2.1-DepartmentDescription by TripType

Figure 2.1 shows a plot between DepartmentDescription and TripType. It is easy to find that some TripType has an obvious relationship with particular Department. For example, Type25 has a highest number of shopping trips in ‘MENS_WEAR’, this triptype maybe mainly related to male shoppers. After investigating some patterns and considered the size of each TripType, I choose Type ‘25’, ‘32’, ‘40’ as my main research objects and set all other TripType to 999 (means ‘others’). Therefore, the response variable only has 4 classes.

b) Missing/Null Value

In train.csv file, there are missing values appear in column ‘Upc’ and ‘FinelineNumber’. Besides, in ‘DepartmentDescription’, there are some records shown as NULL which is meaningless. Therefore, in Excel, I checked these three columns and found that there are only 4,129 out of 647,054 records that have one or more problems. So, to simplify the data cleaning process, I just delete those records.



K27 `=IF(AND(H27=FALSE,I27=FALSE,J27=FALSE),0,1)`

	TripTyp	VisitNumb	Weekday	Upc	DepartmentDescription	FinelineNumb	Check_UPC	Check_Department	Check_Fineline	Check_problem
27	26	8	Friday		NULL		TRUE	TRUE	TRUE	1
550	27	259	Friday		NULL		TRUE	TRUE	TRUE	1
551	27	259	Friday		NULL		TRUE	TRUE	TRUE	1
961	999	409	Friday		NULL		TRUE	TRUE	TRUE	1
1118	39	479	Friday		NULL		TRUE	TRUE	TRUE	1
1136	999	484	Friday		NULL		TRUE	TRUE	TRUE	1
1137	999	484	Friday		NULL		TRUE	TRUE	TRUE	1
1157	44	496	Friday		PHARMACY RX		TRUE	FALSE	TRUE	1
1218	5	521	Friday		PHARMACY RX		TRUE	FALSE	TRUE	1
1375	5	585	Friday		PHARMACY RX		TRUE	FALSE	TRUE	1
1457	5	619	Friday		PHARMACY RX		TRUE	FALSE	TRUE	1
1458	5	619	Friday		PHARMACY RX		TRUE	FALSE	TRUE	1
1459	5	619	Friday		PHARMACY RX		TRUE	FALSE	TRUE	1
1821	5	787	Friday		PHARMACY RX		TRUE	FALSE	TRUE	1
1928	32	845	Friday		NULL		TRUE	TRUE	TRUE	1
1929	32	845	Friday		NULL		TRUE	TRUE	TRUE	1
2033	5	897	Friday		PHARMACY RX		TRUE	FALSE	TRUE	1
2034	5	897	Friday		PHARMACY RX		TRUE	FALSE	TRUE	1
2296	40	1004	Friday		NULL		TRUE	TRUE	TRUE	1
2401	5	1036	Friday		PHARMACY RX		TRUE	FALSE	TRUE	1
2480	5	1064	Friday		PHARMACY RX		TRUE	FALSE	TRUE	1
2568	5	1073	Friday		PHARMACY RX		TRUE	FALSE	TRUE	1
2828	5	1170	Friday		PHARMACY RX		TRUE	FALSE	TRUE	1
2852	40	1179	Friday		NULL		TRUE	TRUE	TRUE	1
3098	44	1236	Friday		NULL		TRUE	TRUE	TRUE	1
3371	9	1355	Friday		PHARMACY RX		TRUE	FALSE	TRUE	1
3493	5	1392	Friday		PHARMACY RX		TRUE	FALSE	TRUE	1
3526	5	1402	Friday		PHARMACY RX		TRUE	FALSE	TRUE	1

Average: 1 Count: 4129 Sum: 4129

Figure 2.2-Check for problematic records

c) Feature Selection

As shown in Form 1.1, not all fields are used in the model. Some are discarded due to multiple reasons:

VisitNumber- an id number. No relationship with TripType.

Upc- an id for product purchased. If used in the model, will results in 93920 dummy variables. The large number of predictors will lead to serious overfitting problem. However, to take full use of the data, I did some experiments on UPC. (See Experiments Section)

3. Data Preprocessing

a) Data Encoding and Dummy Variables

All the category variables need to be treated differently with numeric variables in model. Since weekday and DepartmentDescription are string, they need to be transformed into numbers. I used sklearn.preprocessing.LabelEncoder to finish this work.

```
def Encode_X(X,area):
    for i in area:
        labelencoder_X = LabelEncoder()
        X[:, i] = labelencoder_X.fit_transform(X[:, i])
    return X
```

Then, I used sklearn.preprocessing.OneHotEncoder to create n dummy variables for n classes for each variable.

```python
def create_dummy(X,area):
    onehotencoder = OneHotEncoder(categorical_features = area)
    X = onehotencoder.fit_transform(X)
    return X
```

b) Scaling

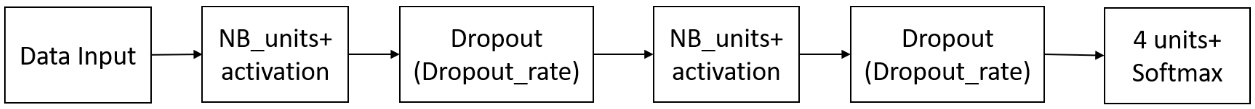
To prevent that some variables with larger values will have larger weights in model, I scale all my features using preprocessing.MaxAbsScaler. The choice of scaler was limited to me because my feature matrix is a sparse matrix (can't be turned into dense due to memory issue). MinMax Scaler is not suitable for sparse matrix, so I choose its similar scaler MaxAbs to scale my data.

c) Data slicing

To test my model, I slice my data into train and test setsDue to the large size of data, it requires much time to tune on test size. So I just set the test size as 0.3 to simplify the process.

Model

1. Overview



2. Model Tuning

To find the best parameters for my model, I used GridSearchCV to tune my model:

```python
def tuning(model,param_grid,x_train,y_train):
    grid = GridSearchCV(estimator=model, param_grid=param_grid)
    grid_result = grid.fit(x_train, y_train,verbose=1)
    return grid_result
```

I made a param_grid with the range of each parameter I want to tune.

```python
neurons = [50,100,150]
dropout_rate = [0.0, 0.1, 0.2]
activation = ['relu','sigmoid']
optimizer = ['SGD', 'RMSprop','Adam']
batch_size = [10, 50, 100]
epochs = [10, 20, 50]
param_grid = dict(batch_size=batch_size, epochs=epochs,
                  optimizer=optimizer,activation=activation,
                  dropout_rate=dropout_rate,neurons=neurons)
```

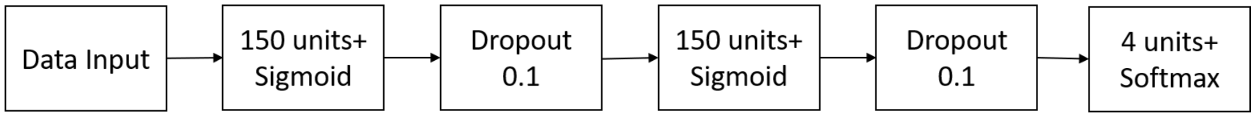
3. Result and Evaluation

a) Loss and Metrics

Since my response variable is a multi-class variable, I choose categorical_crossentropy as loss and set metrics equals to accuracy

b) Result

i. Best model



Batch_size=100,

Epochs=50

ii. Evaluation

Time required for optimization: 0:25:43.586648

Test Accuracy = 0.673721212374

Here is the summarized history of accuracy and loss:

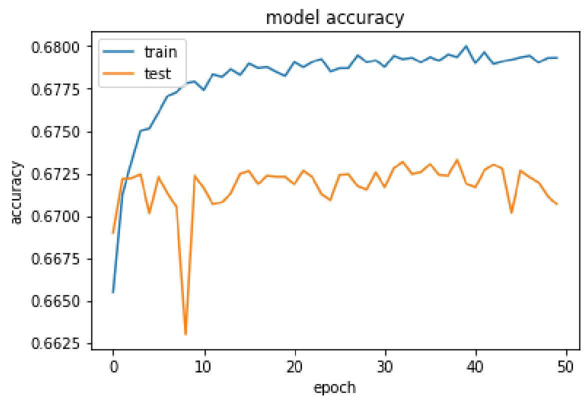


Figure 3.3.1- Summarized history of accuracy

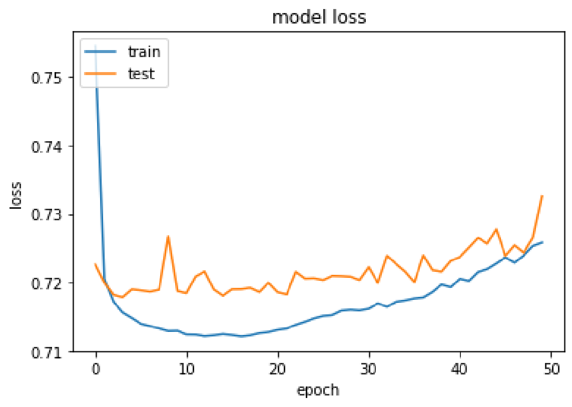


Figure 3.3.2- Summarized history of loss

iii. Conclusion

A model with accuracy of 0.67 normally is not a good one. This might result from the big number of dummy variables, small number of predictors, etc. Also, the large size of data is a great challenge in this assignment.

Experiments

1. UPC

Always want to use as much data as I can, I did some experiments on Upc. Although they all failed in the end, it is good to show my work and thoughts there.

In my data, there is a column 'Upc' which shows the UPC number of the product purchased. I did some research and found that they are not in the standard UPC-A format. In short, UPCs are Universal Product Code that consists of the company code, a product number and a checksum.

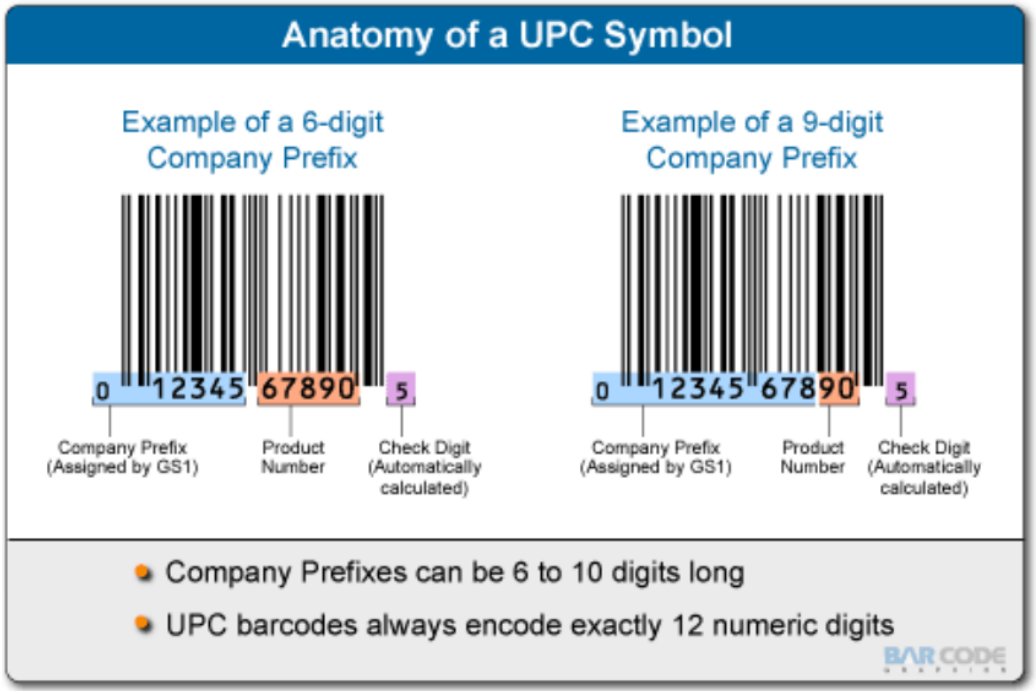
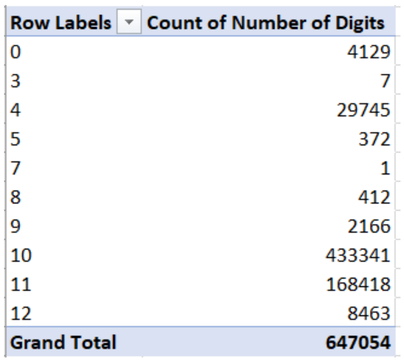

Anatomy of a UPC Symbol
Example of a 6-digit Company Prefix
Example of a 9-digit Company Prefix
0 12345 67890 5
0 12345 67890 5
Company Prefix (Assigned by GS1)
Product Number
Check Digit (Automatically calculated)
Company Prefix (Assigned by GS1)
Product Number
Check Digit (Automatically calculated)
Company Prefixes can be 6 to 10 digits long
UPC barcodes always encode exactly 12 numeric digits
BAR CODE

Figure 4.1-Anatomy of a UPC Symbol

I calculate the number of digits and found the results below:



Row Labels	Count of Number of Digits
0	4129
3	7
4	29745
5	372
7	1
8	412
9	2166
10	433341
11	168418
12	8463
Grand Total	**647054**

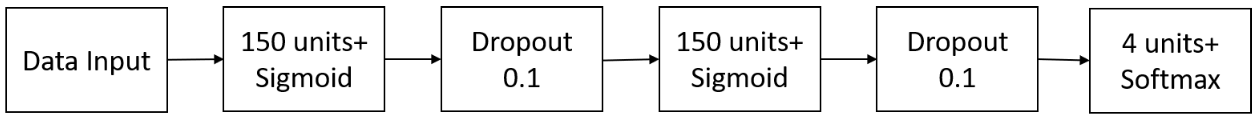
Figure 4.2-Check the number of digits in 'Upc'

I read some posts from Kaggle discussion board and it seemed that most of the participants agreed that most of the UPCs that are less than 12 digits lack the check digit at the end. I don't need to calculate the check digit since my goal is to get the Company_code and Product_code so that I can add variable in my classification.

Therefore, to simplify my analysis, I assume that all Company_code has 6 digits and the rest are Product_code. There are some records with 3,4 and 5 digits, which, to me are some in-house products. So I set their Company_code as 0 and their Upc as their Product_code. For missing values(number of digit=0), I set both variables as -999.

By doing all of the process above, I changed the 'Upc' variable into 'Company_code' and 'Product_code'.

However, again, due to the large number of classes in Company_code and Product_code, they were not used in the model. But it is still possible that if given more information (e.g. the name of Company), I may be able to combine similar classes to reduce the dimentionality.

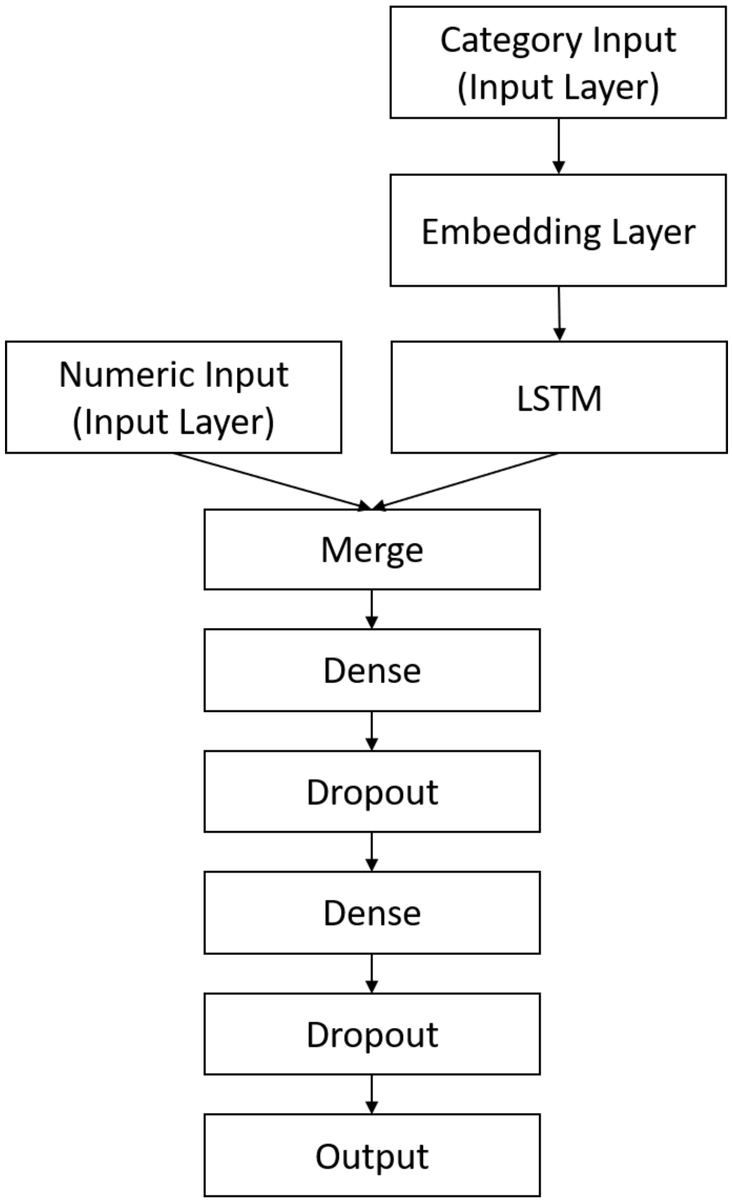


2. Embedding layers

When I met with the problem of too many dummy variables, I searched online and found that Embedding Layers which provided by Keras could give me some help.

According to Keras, embedding layers can be used to turn positive integers (indexes) into dense vectors of fixed size. It is used a lot in text mining where there are many words need to be transformed into numeric values in model. Apparently, it is impractical to use dummy variables.

Therefore, I try to use embedding layers to deal with the category features:

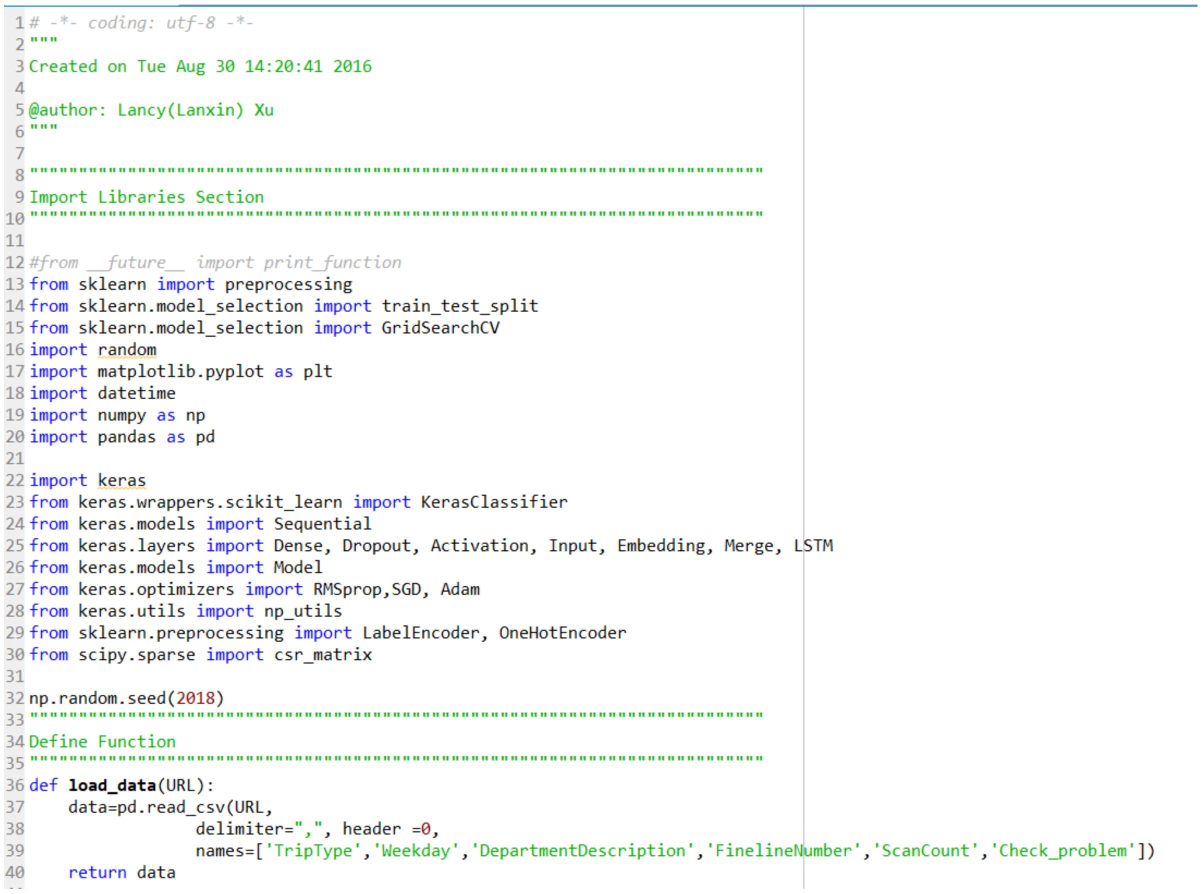


However, the accuracy rate doesn't have significant improvement. Besides, since I don't fully understand the mechanism behind embedding layers, I just decide not to use it as my final model.

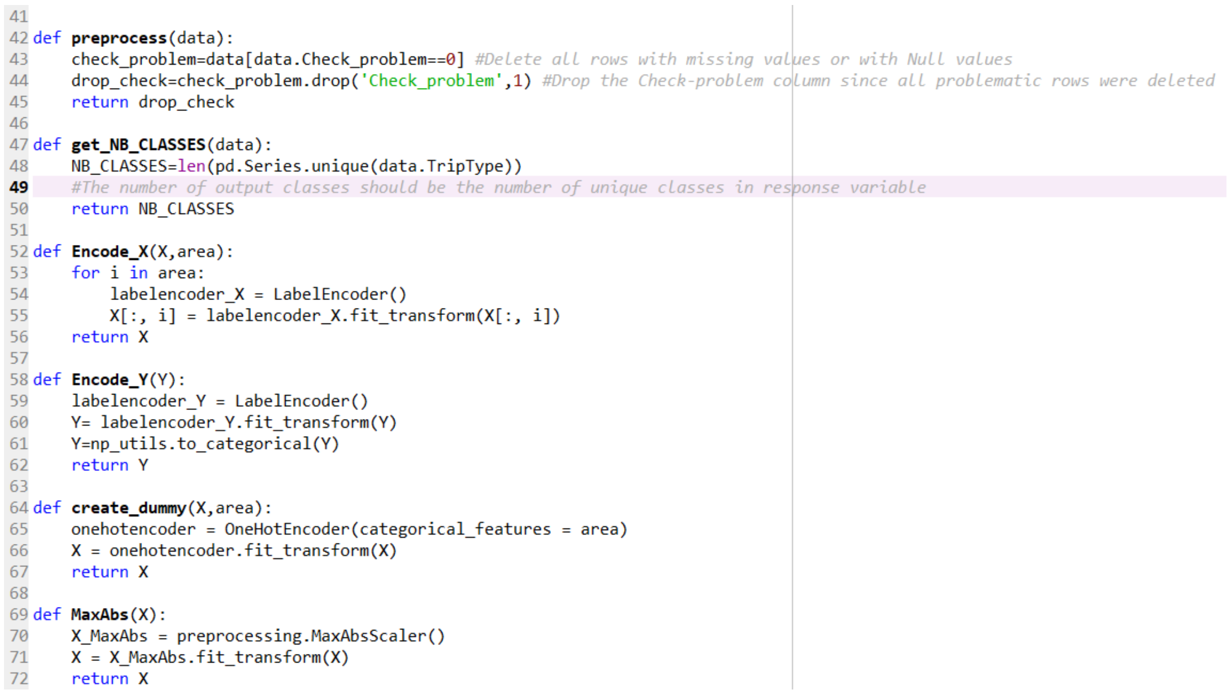
Discussion

Dealing with multi-class features like FinelineNumber(about 6000 classes) or Upc(90k) is really the most difficult part. I tried to turn them all into dummy variables and resulted in a matrix of 50k columns. Using this data to train my model, I got an extremely low accuracy rate of 0.25. This poor performance forced me to discard Upc but also made me think whether there is a way to keep my multi-classes features in my model. Because whenever we train the model, we would like to feed it with more information and take full advantage of the data that we have. As I mentioned before, one way is to embed them to reduce dimensionality. Inspired by text mining, I also came up with an idea of using the techniques in that field to deal with the category variables. After all, the different categories (whether string or numeric) are essentially words. Just as people use word2vec in text mining to reduce dimensionality, we can apply the same principle in category. I am glad to find that many people have the same idea with me. There is already some research and experiments (e.g. a similar method called cat2vec) in this field and will be more improvement in the future. Discussion and further work.

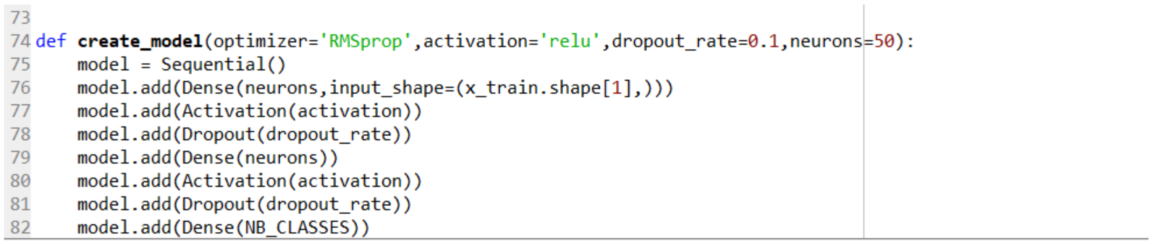
Code



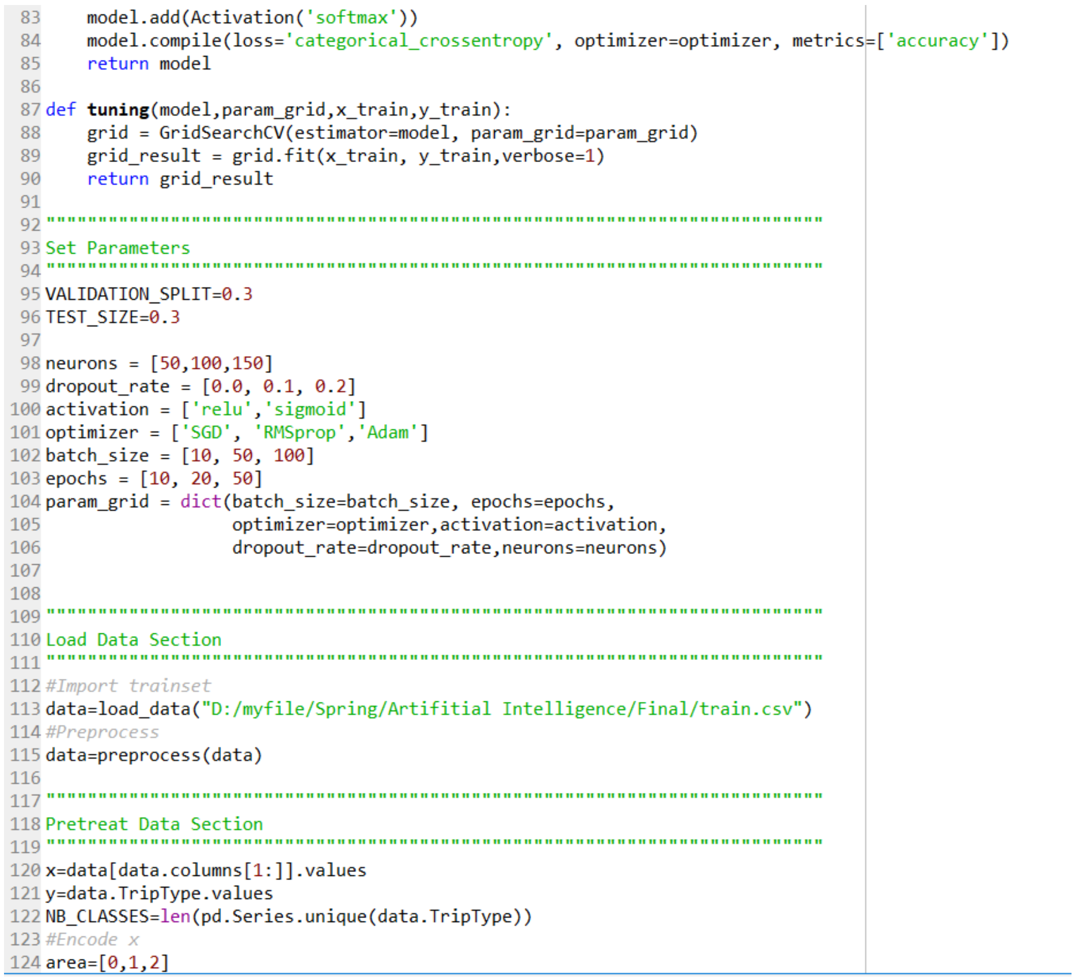
```python
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 30 14:20:41 2016

@author: Lancy(Lanxin) Xu
"""

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Import Libraries Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

#from __future__ import print_function
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import random
import matplotlib.pyplot as plt
import datetime
import numpy as np
import pandas as pd

import keras
from keras.wrappers.scikit_learn import KerasClassifier
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Input, Embedding, Merge, LSTM
from keras.models import Model
from keras.optimizers import RMSprop,SGD, Adam
from keras.utils import np_utils
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from scipy.sparse import csr_matrix

np.random.seed(2018)
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Define Function
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
def load_data(URL):
    data=pd.read_csv(URL,
                 delimiter=",", header =0,
                 names=['TripType','Weekday','DepartmentDescription','FinelineNumber','ScanCount','Check_problem'])
    return data
```



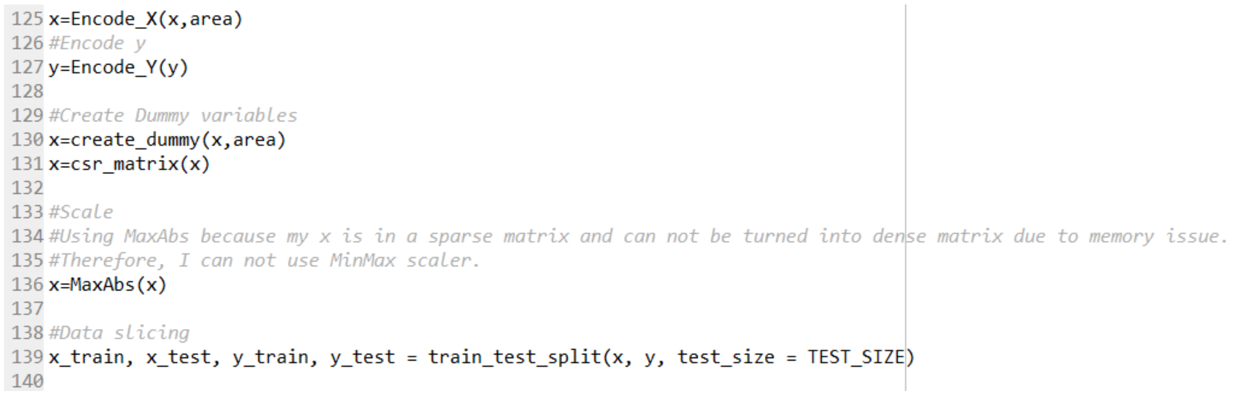
```python

def preprocess(data):
    check_problem=data[data.Check_problem==0] #Delete all rows with missing values or with Null values
    drop_check=check_problem.drop('Check_problem',1) #Drop the Check-problem column since all problematic rows were deleted
    return drop_check

def get_NB_CLASSES(data):
    NB_CLASSES=len(pd.Series.unique(data.TripType))
    #The number of output classes should be the number of unique classes in response variable
    return NB_CLASSES

def Encode_X(X,area):
    for i in area:
        labelencoder_X = LabelEncoder()
        X[:, i] = labelencoder_X.fit_transform(X[:, i])
    return X

def Encode_Y(Y):
    labelencoder_Y = LabelEncoder()
    Y= labelencoder_Y.fit_transform(Y)
    Y=np_utils.to_categorical(Y)
    return Y

def create_dummy(X,area):
    onehotencoder = OneHotEncoder(categorical_features = area)
    X = onehotencoder.fit_transform(X)
    return X

def MaxAbs(X):
    X_MaxAbs = preprocessing.MaxAbsScaler()
    X = X_MaxAbs.fit_transform(X)
    return X
```



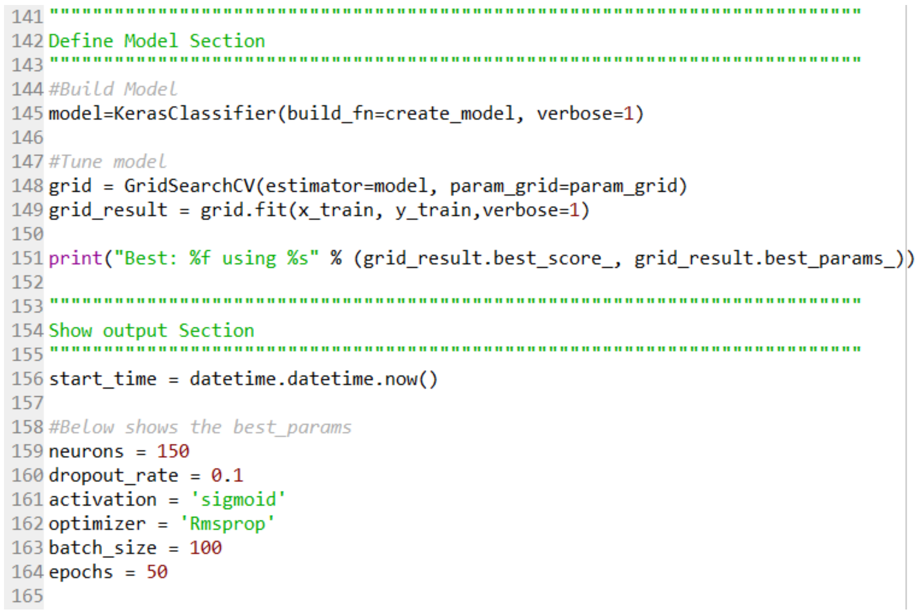
```python

def create_model(optimizer='RMSprop',activation='relu',dropout_rate=0.1,neurons=50):
    model = Sequential()
    model.add(Dense(neurons,input_shape=(x_train.shape[1],)))
    model.add(Activation(activation))
    model.add(Dropout(dropout_rate))
    model.add(Dense(neurons))
    model.add(Activation(activation))
    model.add(Dropout(dropout_rate))
    model.add(Dense(NB_CLASSES))
```



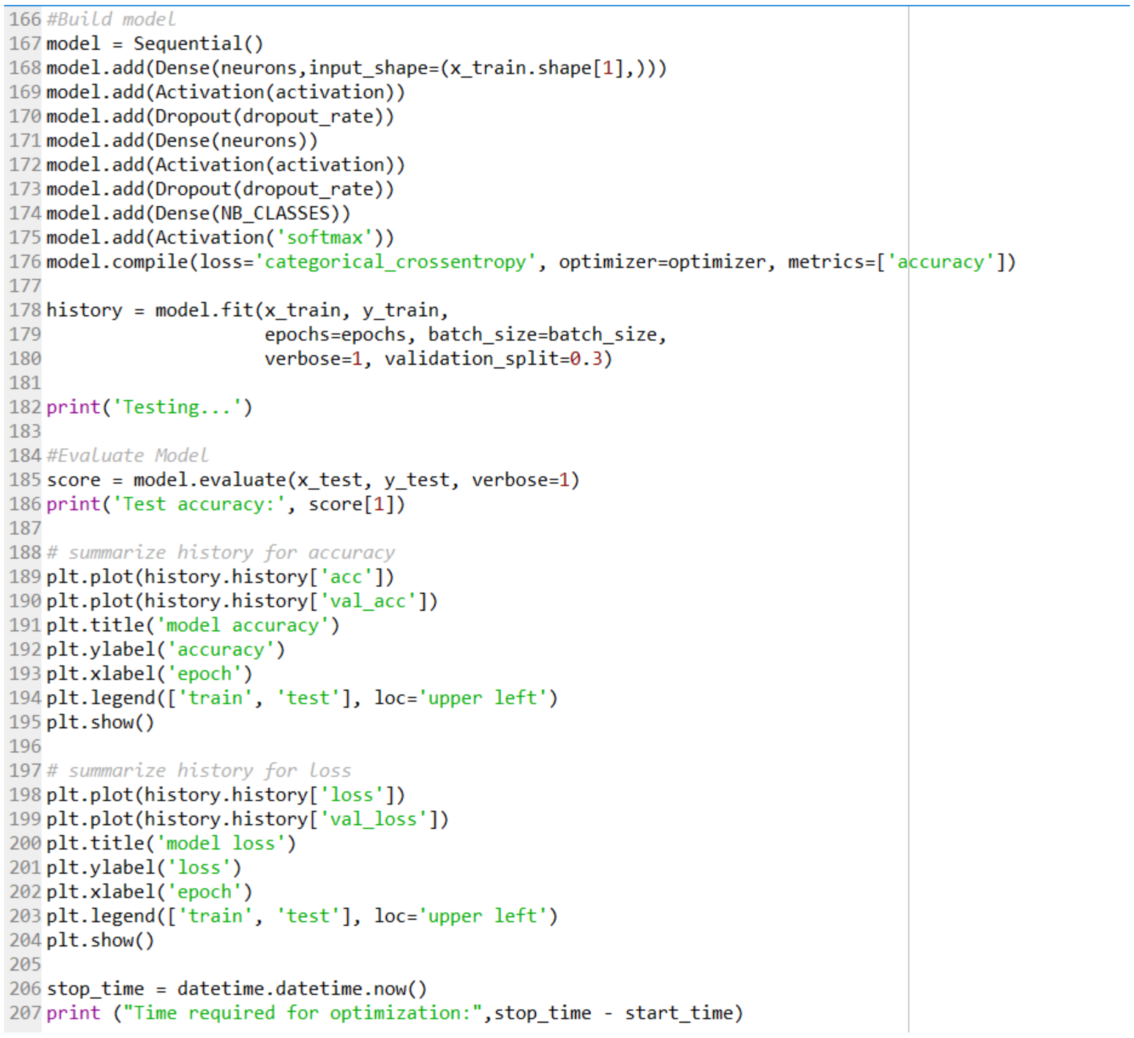
```python
    model.add(Activation('softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model

def tuning(model,param_grid,x_train,y_train):
    grid = GridSearchCV(estimator=model, param_grid=param_grid)
    grid_result = grid.fit(x_train, y_train,verbose=1)
    return grid_result

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Set Parameters
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
VALIDATION_SPLIT=0.3
TEST_SIZE=0.3

neurons = [50,100,150]
dropout_rate = [0.0, 0.1, 0.2]
activation = ['relu','sigmoid']
optimizer = ['SGD', 'RMSprop','Adam']
batch_size = [10, 50, 100]
epochs = [10, 20, 50]
param_grid = dict(batch_size=batch_size, epochs=epochs,
                  optimizer=optimizer,activation=activation,
                  dropout_rate=dropout_rate,neurons=neurons)


""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Load Data Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#Import trainset
data=load_data("D:/myfile/Spring/Artifitial Intelligence/Final/train.csv")
#Preprocess
data=preprocess(data)

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Pretreat Data Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
x=data[data.columns[1:]].values
y=data.TripType.values
NB_CLASSES=len(pd.Series.unique(data.TripType))
#Encode x
area=[0,1,2]
```



```python
x=Encode_X(x,area)
#Encode y
y=Encode_Y(y)

#Create Dummy variables
x=create_dummy(x,area)
x=csr_matrix(x)

#Scale
#Using MaxAbs because my x is in a sparse matrix and can not be turned into dense matrix due to memory issue.
#Therefore, I can not use MinMax scaler.
x=MaxAbs(x)

#Data slicing
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = TEST_SIZE)

```



```python
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Define Model Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#Build Model
model=KerasClassifier(build_fn=create_model, verbose=1)

#Tune model
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_result = grid.fit(x_train, y_train,verbose=1)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Show output Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
start_time = datetime.datetime.now()

#Below shows the best_params
neurons = 150
dropout_rate = 0.1
activation = 'sigmoid'
optimizer = 'Rmsprop'
batch_size = 100
epochs = 50

```



```python
#Build model
model = Sequential()
model.add(Dense(neurons,input_shape=(x_train.shape[1],)))
model.add(Activation(activation))
model.add(Dropout(dropout_rate))
model.add(Dense(neurons))
model.add(Activation(activation))
model.add(Dropout(dropout_rate))
model.add(Dense(NB_CLASSES))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    epochs=epochs, batch_size=batch_size,
                    verbose=1, validation_split=0.3)

print('Testing...')

#Evaluate Model
score = model.evaluate(x_test, y_test, verbose=1)
print('Test accuracy:', score[1])

# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

stop_time = datetime.datetime.now()
print ("Time required for optimization:",stop_time - start_time)
```