Authors: Bader Eddin Loukil, Nevio Roccia

San Francisco Crime Data Project

This Notebook contains code for a neural network aimed at predicting crime data from the "San Francisco Crime Classification" competition on kaggle. The authors decided on computing a neural network, since it can handle classification cases.

Pre-Processing the data

In [1]:
```
from datetime import datetime
import pandas as pd
import numpy as np
from tensorflow import keras
%matplotlib inline
import matplotlib.pyplot as plt
```

For an additional category to be used in our classification we decided to import an unemployment rate dataset. Unemployment has been proven to be a strong predictor of crime, which is why we think it would be a great addition for this competition. We got this dataset from here: https://labormarketinfo.edd.ca.gov/cgi/databrowsing/localAreaProfileQSMoreResult.asp?menuChoice=localAreaPro&criteria=unemployment+rate&categoryType=employment&geogArea=0604000075&area=San+Francisco+Count

In [2]:
```
 #Importing an unemployment rate dataset for San Francisco
extra_data=pd.read_csv("unemployment rate20221259.csv")
#Selecting the relevant columns from the dataset
extra_data=extra_data[["Year", "Period","Unemployment Rate %"]]
#Replacing the month string with integers
d={"Jan": 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5, 'Jun': 6, 'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10, 'Nov': 
extra_data.Period = extra_data.Period.map(d)
#Renaming the column
extra_data= extra_data.rename(columns={'Period': 'Month'})
```

In [3]:
```
extra_data.head()
```

Out[3]:

	Year	Month	Unemployment Rate %
0	2022	1.0	3.5
1	2022	2.0	3.0
2	2022	3.0	2.5
3	2022	4.0	2.2
4	2022	5.0	1.9

In [4]:
```
#Reading the csv Files containing the crime data
train_data=pd.read_csv("train.csv")
test_data=pd.read_csv("test.csv")
```

In the following code cell we transformed the date column into the "date" variable type and then split the date column into multiple columns, so it would be easier to work with. The Month column might be interesting since there is evidence, that crime happens more often around Christmas for example when people don't have enough money for presents. The same goes for the hour since you'd expect more breakins for example during the night.

In [5]:
```
#Processing Train Data
train_data.drop_duplicates(keep = 'first', inplace = True)
dates = pd.to_datetime(train_data["Dates"])
train_data["Year"]= dates.dt.year
train_data["Month"] = dates.dt.month
train_data["Day"] = dates.dt.day
train_data["Hour"] = dates.dt.hour
train_data["Minutes"] = dates.dt.minute
train_data["DayOfWeek"]= dates.dt.weekday
```

In [6]:
```
#Merging the train data with our extra_data dataset
train_data=pd.merge(train_data, extra_data, on=["Year","Month"])
```

In [7]:
```
#Splitting the PdDistrict Column into multiple columns, one for each district
dummies_train=pd.get_dummies(train_data['PdDistrict'])
train_data=pd.concat([train_data,dummies_train],axis=1)
```

In [8]:
```
#Deleting 5 unnecessary columns
train_data=train_data.drop(columns=["Descript", "Resolution", "Address","Dates","PdDistrict"])
train_data.head()
```

Out[8]:

	Category	DayOfWeek	X	Y	Year	Month	Day	Hour	Minutes	Unemployment Rate %	BAYVIEW	CENTRAL	INGLESIDI
0	WARRANTS	2	-122.425892	37.774599	2015	5	13	23	53	3.6	0	0	(
1	OTHER OFFENSES	2	-122.425892	37.774599	2015	5	13	23	53	3.6	0	0	(
2	OTHER OFFENSES	2	-122.424363	37.800414	2015	5	13	23	33	3.6	0	0	(
3	LARCENY/THEFT	2	-122.426995	37.800873	2015	5	13	23	30	3.6	0	0	(
4	LARCENY/THEFT	2	-122.438738	37.771541	2015	5	13	23	30	3.6	0	0	(

The above table shows our final processed train data.

Here we applied the same processing steps on our test data, so we would end up with two homogenous datasets.

In [9]:
```
#Processing test_data
test_data.drop_duplicates(keep = 'first', inplace = True)
dates1 = pd.to_datetime(test_data["Dates"])
test_data["Year"]= dates1.dt.year
test_data["Month"] = dates1.dt.month
test_data["Day"] = dates1.dt.day
test_data["Hour"] = dates1.dt.hour
test_data["Minutes"] = dates1.dt.minute
test_data["DayOfWeek"]= dates1.dt.weekday
```

In [10]:
```
#Splitting the PdDistrict Column into multiple columns, one for each district
dummies_test=pd.get_dummies(test_data['PdDistrict'])
test_data=pd.concat([test_data,dummies_test],axis=1)
```

In [11]:
```
#Deleting 3 unnecessary columns
test_data=test_data.drop(columns=["Address","Dates","PdDistrict"])
```

In [12]:
```
# Adding Unemployment rate to the test_data
test_data=pd.merge(test_data, extra_data, on=["Year","Month"])
```

In [13]:
```
#Creating new dataset from the train data minus the Category column
X=train_data.drop(["Category"],axis=1)
X=X.astype(float)
```

In [14]:
```
#Creating a new dataset from the previously dropped column "Category"
y=pd.get_dummies(train_data["Category"])
y=y.astype(float)
```

In [15]:
```
#Importing the method for splitting data into train and test data
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X.values,y.values,test_size=0.1,random_state=42)
```

The following code describes our neural network. We worked with the "relu" activation for the hidden layers and the "softmax" activation for the output layer. We chose these hyperparameters because we know from our experiences from labs we did in class, that this combination works best for classification problems. After a bit of trial and error with other hyperparameters we confirmed that this is the case here as well. We decided to go with less hidden layers but them larger than in other trials, because this seems to yield better results. Finally, the output layer has 39 nodes since we are trying to classify for 39 different categories.

In [16]:
```
#Building a neural network
from keras.layers import Dense
from keras import Sequential
model=Sequential()
model.add(Dense(265, activation = "relu", input_shape=(X.shape[1],)))
model.add(Dense(128,activation='relu'))
model.add(Dense(60,activation='relu'))
#model.add(Dense(30,activation='relu'))
#Adding an output layer with 39 nodes, one for each category
model.add(Dense(39,activation='softmax'))
```

We compile the model with the "adam" optimizer, the "categorial crossentropy" loss function and the accuracy metric, which is pretty standard for classifications.

In [17]:
```
#Compiling the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

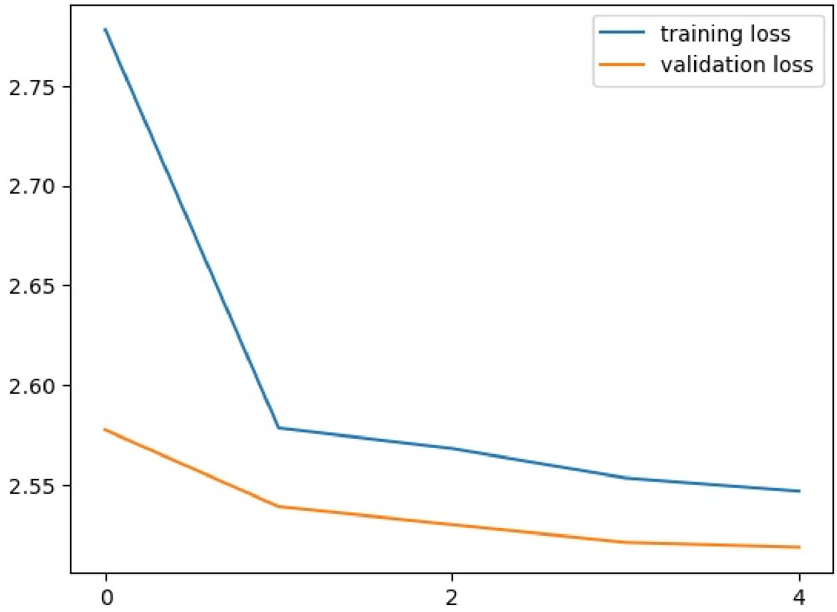
In [18]:
```
#Fitting the model
history=model.fit(X_train,y_train,
         batch_size=40,
```

```
         epochs=5,
         validation_split = 0.01)
```

```
Epoch 1/5
19507/19507 [==============================] - 36s 2ms/step - loss: 2.7780 - accuracy: 0.2091 - val_loss: 2.577
6 - val_accuracy: 0.2248
Epoch 2/5
19507/19507 [==============================] - 40s 2ms/step - loss: 2.5786 - accuracy: 0.2351 - val_loss: 2.539
2 - val_accuracy: 0.2501
Epoch 3/5
19507/19507 [==============================] - 46s 2ms/step - loss: 2.5683 - accuracy: 0.2379 - val_loss: 2.530
2 - val_accuracy: 0.2504
Epoch 4/5
19507/19507 [==============================] - 47s 2ms/step - loss: 2.5536 - accuracy: 0.2402 - val_loss: 2.521
3 - val_accuracy: 0.2504
Epoch 5/5
19507/19507 [==============================] - 47s 2ms/step - loss: 2.5469 - accuracy: 0.2400 - val_loss: 2.518
9 - val_accuracy: 0.2460
```

In [19]:
```
#Plotting the training and validation loss
def plot_history(history: keras.callbacks.History):
  n = len(history.history['loss'])
  plt.plot(np.arange(n), history.history['loss'], label="training loss")
  plt.plot(np.arange(n), history.history['val_loss'], label="validation loss")
  plt.xticks(range(0, n + 1, 2))
  plt.legend()
  plt.show()

plot_history(history)
```



In [20]:
```
#Preparing the test data for testing
Xtest=test_data.drop(["Id"],axis=1)
```

In [21]:
```
#Using our model to predict the data
pred=model.predict(Xtest.values)
```

```
27634/27634 [==============================] - 44s 2ms/step
```

These last few code cells contain the code for the submission in the competition on kaggle.

In [22]:
```
m = np.max(pred, axis=1).reshape(-1, 1)
predicted = np.array((pred == m), dtype='int32')
pred
```

Out[22]:
```
array([[1.67128320e-07, 7.39302812e-03, 4.62950567e-10, ...,
        2.63333022e-05, 9.79841128e-02, 5.38809167e-04],
       [2.01449637e-07, 7.83872884e-03, 5.21567900e-10, ...,
        2.98132127e-05, 9.93108302e-02, 5.85741655e-04],
       [1.74540489e-07, 7.36579206e-03, 3.60316221e-10, ...,
        2.67686319e-05, 9.65273380e-02, 5.56318962e-04],
       ...,
       [5.49432895e-11, 3.68526060e-04, 5.95997708e-15, ...,
        9.07765667e-08, 3.13008986e-02, 1.55928083e-05],
       [8.08265330e-11, 3.92288639e-04, 2.36050485e-15, ...,
        1.11278027e-07, 3.01331263e-02, 2.00147333e-05],
       [1.25145476e-11, 2.05785429e-04, 5.70136394e-16, ...,
        3.12732205e-08, 2.44393628e-02, 8.07276865e-06]], dtype=float32)
```

In [23]:
```
#Reading the sampleSubmission csv file
```

```python
sample_submission = pd.read_csv("sampleSubmission.csv")
col_names=list(sample_submission.columns)
col_names.remove('Id')
```

In [24]:
```python
#Comparing the sample Submission file to our predicted test data
submission = pd.DataFrame()
submission['Id'] = test_data['Id']
for i , entry in enumerate(col_names):
    submission[entry] = predicted[:,i]
```

In [25]:
```python
submission.to_csv('submission.csv', index=False)
```

The final score we were able to achieve on kaggle was: 28.2612