Artificial Intelligence - MSc

CS6501 - MACHINE LEARNING APPLICATIONS

Instructor: Enrique Naredo

CS6501_Kaggle

Current Date

Today: 2021 / 11 / 28

Show code

Enter your details here:

Team_Number: " 14 "

Team_Name: " TeamFourteen "

Student_ID: " 21183147 "

Student_full_name: " Sarthak Punjabi "

Student_ID: " 21006415 "

Student_full_name: " Shagil chaudhary "

Student_ID: " 21041784 "

Student_full_name: " humraj singh sorout "

Student_ID: " 21143838 "

Student_full_name: " Anupriya Shanmugam "

Student_ID: " Insert text here "

Student_full_name: " Insert text here "

Show code

```
#@title Notebook information
Notebook_type = 'Assignment' #@param ["Example", "Lab", "Practice", "Etivity", "
Version = "Final" #@param ["Draft", "Final"] {type:"raw"}
Submission = True #@param {type:"boolean"}
```

Notebook information

Notebook_type: Assignment

Version: Final

Submission: ✓

INTRODUCTION

The aim of this experiment is to design and develop a system to predict the price of a house based on features given in the dataset. The predictions from this system will simultaneously be submitted to a ranked Kaggle competition in addition to the usual submission on SULIS. We are expected to implement the machine learning concepts studied during this week, i.e. Natural Language Processing and Long Short Term Memory. We are also presented with the opportunity to apply prior knowledge to design an effective model.

Imports

```
import numpy as np
import pandas as pd
from scipy.sparse import hstack
import folium
import matplotlib.pyplot as plt
import seaborn as sns
from folium.plugins import MarkerCluster
import nltk
import re
import string
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LinearRegression
from xgboost import XGBRegressor
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
```

```
# Import data
df = pd.read_csv("HousePrice_Train.csv")
sns.set_theme(style="whitegrid",
              palette="colorblind")
```

```
# View data
df
```

the description of data shows that it has a mix of textual, numerical, and categorical data.

Exploring the dataset.

```
df.info()
```

```
df.describe()
```

The describe() function gives us the statistical properties of the data. It gives us stats about numerical data only.

```
df.shape
```

```
df.head(5) # First five rows
```

Here, we look for the null values present in the dataset.

```
df.isnull().sum() # Check number of missing values per column
```

Here, BER_class and Services columns have several missing values that has to be dealt with. We plot histogram for few specific columns to better understand the data by visualisation.

```
# Draw histograms for data
column_list = ['Num_Bathrooms', 'Num_Beds', 'BER_class', 'Type','Surface', 'Pric
for column in column_list:
    plt.figure(figsize=(20,5))
    sns.histplot(df[column]) # Create histogram
    plt.show() # Show histogram
```

Scatter plots are used for the examination of the relationship between the predictor variable and the target variable.

```
# scatterplot between numerical variables and price
column_list = ['Num_Bathrooms', 'Num_Beds', 'Latitude', 'Longitude','Surface']
for column in column_list:
    plt.figure(figsize=(20,5))
    sns.scatterplot(df[column],df["Price"])
    plt.show()
```

following is observed through the scatterplots:

- There is a slight correlation between number of bathrooms and price
- The same can be said for number of bedrooms.
- There's a very slight correlation between location and price
- There is a strong positive correlation between price and surface area. Not surprising.

```
# Box plots are used for the purpose of detecting the outliers.
column_list = ['BER_class','Type']
for column in column_list:
    plt.figure(figsize=(20,5))
    sns.boxplot(df[column],df["Price"])
    plt.show()
```

- The box plots for BER class showcases quite a few outliers.
- There are plenty of outliers for the “detached” home too.

We calculate a correlation matrix and plot it using a heatmap.

```
plt.figure(figsize= (10, 10)) # Define plot size, increase if the graph is crowd
sns.heatmap(df.iloc[:,2:].corr(),square=True, annot=True)
```

The correlation heatmap shows that price is highly correlated to the number of bedrooms and bathrooms. This could be caused due to outliers. This is why we observe the relationship between variables using scatter plots.

The “folium” package allows us to visualize a set of coordinates on an interactive map on interactive python notebooks, so we use it here to draw up a map of our houses. This should also help us investigate the outlier we observed earlier.

```
locations = df[['Latitude', 'Longitude']]
locationlist = locations.values.tolist() # converting to a list

# Set map parameters.
map = folium.Map(location=[df['Latitude'].median(), df['Longitude'].median()], #
                 zoom_start=12)
marker_cluster = MarkerCluster().add_to(map)
for point in range(0, len(locationlist)): # Iterate through list of coordinates
    folium.Marker(locationlist[point], popup=df['Location'][point]).add_to(marke
map # Display the map. Takes a while to load!
```

Most of the data is in and around Dublin. However, a singular point in the United Kingdom. Considering we're looking at Ireland housing data, it's safe to say this point was included in our dataset accidently.

Preprocessing

Removing outliers

The first thing we do is remove the outlier we just spotted in the data exploration.

```
df[df["Latitude"] < 52.6]
```

```
df[df["Longitude"] > -2]
```

```
outlier_index = df[df["ID"] == 12270559].index # Store index of row where ID = 1
df.drop(outlier_index, inplace=True) # Delete it
```

We can verify that this point is gone by searching for the same ID again.

```
df[df["ID"] == 12270559] # find rows where ID = 12270559
df[df["Num_Bathrooms"] > 10]
df[df["Num_Beds"] > 10]
fancy_house_IDs = ["12381836","11780612","12085770"]
for houseID in fancy_house_IDs:
    outlier_index = df[df["ID"] == houseID].index
    df.drop(outlier_index, inplace=True)
```

We can also directly check the condition and drop the outliers by index as such:

```
outlier_index = df[df["Surface"] > 5000].index
df.drop(outlier_index, inplace=True)
```

The above code gets rid of 2 houses with an oddly high surface area.

We can validate our oulier removal by running the plots again. Let's check the heatmap again.

```
plt.figure(figsize= (10, 10))
sns.heatmap(df.iloc[:,2:].corr(),square=True, annot=True)
```

Numerical and categorical data

Initially we use a subset of the original dataframe containing only numerical and categorical data.

```
df_numeric = df[["Location","Num_Bathrooms","Num_Beds","BER_class","Latitude","L
```

In order to apply machine learning algorithms to categorical data, we must transform it using the naive approach known as “encoding”, specifically “one-hot encoding”.

```
# Initialize (set up) encoder to do the heavy lifting for us
encoder = OneHotEncoder(drop="first", # Remove the first column
                        sparse=False, # Set to return an array instead of a matr
                        handle_unknown="ignore") # Don't throw errors up if null
```

```
columns_to_replace = ["Location","BER_class","Type"] # List out columns to encod
encoded_data = encoder.fit_transform(df[columns_to_replace]) # Encode them
encoded_data = pd.DataFrame(encoded_data) # Convert this to a dataframe
encoded_data.head() # View data
```

```
encoded_data.columns = encoder.get_feature_names_out() # Rename encoded columns
```

```
encoded_data.head()
```

Here, we only get the encoded data after this the next step is to drop the original columns and join the encoded data with the actual data.

```
for column in columns_to_replace: # Iterate through list of columns
    df_numeric.drop(column ,axis=1, inplace=True) # Drop (delete) the ones we en
df_numeric = encoded_data.join(df_numeric) # Concatenate (join) the dataframes
df_numeric.columns.tolist() # Check column names of final encoded data
```

After getting all the data together the next step is to scale the data to make the model computationally efficient.

```
df_numeric_noscale = df_numeric.copy()
scaler = MinMaxScaler() # Initialize scaler
for column in df_numeric.columns:
    df_numeric[column]=pd.DataFrame(scaler.fit_transform(df_numeric[column].valu
```

We can, observe the results of scaling by using the describe() function again.

```
df_numeric.describe()
```

Textual data

```
df_textual = df[["Description","Services","Features","Price"]]
```

We transform words into numbers to feed them into our traditional machine learning algorithms. A major difference here is that we initially have to clean the data and extract the most significant words, before assigning values to them, a process known as “vectorizing”.

To start off, we initialize our lemmatizer object.

We also download a few sets:

- Wordnet: the Lemmatization dataset.
- Stopwords: Common “stop” words that do not add much to the context of a sentence.
- Punkt: Punctuation marks.

```
lemmatizer = WordNetLemmatizer() # Initialize Lemmatizer
nltk.download('wordnet')# Download lemmatization database
nltk.download('stopwords') # Download list of stop words
nltk.download('punkt') # download list of punctuation symbols
stopwords_set = set(stopwords.words('english')) # Define list of stop words
```

Shows us the list of stop words.

```
stopwords_set
```

We now define a cleaning function that should be able to reduce entire sentences down to their base forms.

```
def clean(row):
    input_data = str(row)

    lowertext = input_data.lower() # convert to lower case
    tokens = word_tokenize(lowertext) # Tokenize
    df_stopwords=[word for word in tokens if word not in stopwords_set] # Remove
    df_punctuation=[re.sub(r'['+string.punctuation+']+', ' ', i) for i in df_sto
    df_whitespace = ' '.join(df_punctuation) # Remove multiple whitespace
    lemmatizer = WordNetLemmatizer() # Initialize lemmatizer
    df_lemma = lemmatizer.lemmatize(df_whitespace) # Lemmatize
    df_lemma_tokenized = word_tokenize(df_lemma) # Tokenize again
    df_lemma_shortwords = [re.sub(r'^\w\w?$', '', i) for i in df_lemma_tokenized
    df_lemma_whitespace =' '.join(df_lemma_shortwords) # Join whitespace again
    df_lemma_multiplewhitespace = re.sub(r'\s\s+', ' ', df_lemma_whitespace) # J
    df_clean = df_lemma_multiplewhitespace.lstrip(' ') #Remove any whitespace at

    return df_clean
```

Taking it step by step:

- Lowercase: easier to process everything when you don’t have to worry about case sensitivity.
- Tokenize: Split sentences into words, with each word being a “token”.
- Remove stopwords: Scans through the tokens of each sentence, checks them against the big list of stopwords, and only keeps them if the aren’t present in the stopwords list.
- Remove punctuation: Punctuation usually doesn’t add much to the context of a sentence, and can be very ambiguous depending on usage. It’s better to remove them altogether.
- Whitespace: A lot of these operations don’t actually remove the offending characters, but rather replace them with a space. We join multiple spaces together quite often here.
- Lemmatization: Reduces words to their base form.

```
df_textual["Description"][0] # Original
clean(df_textual["Description"][0]) # Cleaned
df_textual_cleaned = df_textual.iloc[:,:-1].applymap(clean) # Apply cleaning to
df_textual_cleaned = pd.concat([df_textual_cleaned,df_textual["Price"]],axis=1)
df_textual_cleaned.head() # View cleaned dataset
```

The next step is to vectorize the data and apply to the model.

NLP

Vectorization

The TF-IDF for each term (word) is calculated by

$$tf * idf$$

Where

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

$$idf(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

Complex formulae aside, the term frequency(TF) is simply the number of times a term(word) occurs in a document(sentence). Inverse Document Frequency(IDF) is the logarithmic inverse of the fraction of documents that contain the term. (meaning, divide total number of sentences by the number of sentences containing that word, and then take the log).

```
# Initialize vectorizer
vectorizer = TfidfVectorizer(max_features= 500, # consider only the top 500 comm
                             max_df=0.7) # Ignore words that appear in more than
#we first apply the vectorizer to each column separately.
X0 = vectorizer.fit_transform(df_textual_cleaned.iloc[:,0])
X1 = vectorizer.fit_transform(df_textual_cleaned.iloc[:,1])
X2 = vectorizer.fit_transform(df_textual_cleaned.iloc[:,2])
```

```
#then we combine them.
X_textual = hstack((X0, X1, X2))
X_textual.shape
```

```
y_textual = df_textual_cleaned.iloc[:,-1]
y_textual.shape
```

Modelling

```
X_textual_train, X_textual_val, y_textual_train, y_textual_val = train_test_spli
```

```
# create a model
MNB = MultinomialNB()

# fit to data
MNB.fit(X_textual_train, y_textual_train)
```

```
# Training accuracy
y_textual_train_pred = MNB.predict(X_textual_train)
accuracy_score(y_textual_train, y_textual_train_pred)
```

```
# Validation accuracy
y_textual_val_pred = MNB.predict(X_textual_val)
accuracy_score(y_textual_val, y_textual_val_pred)
```

Predicting on test data

In order to predict data, we must import the given test data and apply the same operations we did on our training data in order for our model to recognize it. In this case, we have to clean and vectorize our data.

```
df_test = pd.read_csv("HousePrice_Test.csv")
df_textual_test = df_test[["Description","Services","Features"]]
df_textual_cleaned_test = df_textual_test.iloc[:,:-1].applymap(clean) # Apply cl
X0_test = vectorizer.fit_transform(df_textual_cleaned.iloc[:,0])
X1_test = vectorizer.fit_transform(df_textual_cleaned.iloc[:,1])
X2_test = vectorizer.fit_transform(df_textual_cleaned.iloc[:,2])
X_textual_test = hstack((X0_test, X1_test, X2_test))
y_pred = MNB.predict(X_textual_test)
```

For the Kaggle competition, we are required to submit our predictions in a specific format. The following code takes our predictions and creates the file for us.

```
# save predictions in a file
df_id = pd.DataFrame(data=np.arange(1639,2341), columns = ['Index'])
df_class = pd.DataFrame(data=y_pred, columns = ['Price'])
df_pred = pd.concat([df_id, df_class], axis=1)

# change 'YourTeam' by your team number, for instance: Team-1
df_pred.to_csv('CS6501_Kaggle_TeamFourteen.csv', sep=',', index=False)
```

LSTM

Modelling

"In general, with neural networks, it’s safe to input missing values as 0, with the condition that 0 isn’t already a meaningful value. The network will learn from exposure to the data that the value 0 means missing data and will start ignoring the value."

Since our data is scaled from 0 to 1, we impute(replace) the missing values with -1.

```
df_numeric = df_numeric.fillna(-1)
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call
last)
<ipython-input-1-7fd2fa859b44> in <module>()
----> 1 df_numeric = df_numeric.fillna(-1)

NameError: name 'df_numeric' is not defined
```

```
X_numeric = df_numeric.iloc[:,:-1]
X_numeric.shape
```

```
y_numeric = df_numeric.iloc[:,-1]
y_numeric.shape
```

```
X_numeric_train, X_numeric_val, y_numeric_train, y_numeric_val = train_test_spli
```

Since neural networks work with multiple layers, we have to define each layer:

```
regressor = Sequential()

num_units = 300 # Number of units per layer
drop_value = 0.65 # Change to drop out connections

# Add LSTM layer
regressor.add(LSTM(units = num_units,
                   return_sequences = True,
                   input_shape = (X_numeric_train.shape[1], 1)))

# Add dropout layer
regressor.add(Dropout(drop_value))

regressor.add(LSTM(units = num_units, return_sequences = True))
regressor.add(Dropout(drop_value))

regressor.add(LSTM(units = num_units, return_sequences = True))
regressor.add(Dropout(drop_value))

regressor.add(LSTM(units = num_units))
regressor.add(Dropout(drop_value))

# Add dense layer for output predictions
regressor.add(Dense(units = 1))
```

While comipling we specify an optimization algorithm, a loss metric to minimize, and a performance metric to measure and report.

```
regressor.compile(optimizer='adam', # Configures the model for training using "A
              loss='mean_squared_error', # Loss function
              metrics=['accuracy']) # Performance metric
```

In order to make training more efficient, we can define “callbacks”.

Following callbacks are implemented:

- ModelCheckpoint – Saves the best model (whichever has the highest metric-accuracy in our case)
- EarlyStopping – Stops the model if performance does not improve for a certain number of epochs
- ReduceLROnPlateau – Reduces the learning rate if performance stagnates.

```
# Define callbacks
checkpoint = ModelCheckpoint("checkpoints", # Directory
                             monitor='accuracy', # Performance metric to monitor
                             verbose=1, # Print update messages
                             save_best_only=True, # Save only best performing mo
                             save_weights_only=False, # Save only weights from m
                             mode='max', # Criteria to replace saved model
                             save_freq='epoch') # Frequency to save model

earlystop = EarlyStopping(monitor='accuracy',
                          min_delta=1e-4, # Minimum change in the monitored quan
                          patience=7, # Number of epochs with no improvement
                          verbose=1,
                          mode='max',
                          baseline=None, # Baseline value for the monitored quan
                          restore_best_weights=True) # restore model weights fro

lrreduction = ReduceLROnPlateau(monitor='accuracy',
                                factor=0.01, # new lr = lr * factor.
                                patience = 4, # number of epochs with no improve
                                verbose=1,
                                mode='max',
                                min_delta=1e-4, # threshold for measuring the ne
                                cooldown=0, # number of epochs to wait before re
                                min_lr=1e-6) # lower bound on the learning rate

callbacks = [checkpoint, earlystop, lrreduction]
```

```
num_epochs = 128
regressor.fit(X_numeric_train, y_numeric_train,
              epochs = num_epochs,
              batch_size = 32,
              callbacks = callbacks)
```

We load the "best" model defined from our callbacks:

```
best_model = keras.models.load_model('checkpoints')
```

```
# Training set mean squared error (MSE)
y_numeric_train_pred = best_model.predict(X_numeric_train)
```

```
y_numeric_train_pred = scaler.inverse_transform(y_numeric_train_pred)
```

```
# Validation MSE
y_numeric_val_pred = best_model.predict(X_numeric_val)
```

```
y_numeric_val_pred = scaler.inverse_transform(y_numeric_val_pred)
```

```
mean_squared_error(y_numeric_val, y_numeric_val_pred)
```

Predicting on test data

A key detail to remember is that while the encoder is fit on training data and applied to test data with transform(), the scaler must be fit to training data too. We must also remember to use the scaler to inverse transform the data in order to get non-scaled predictions.

```
df_test = pd.read_csv("HousePrice_Test.csv")
```

```
df_numeric_test = df_test[["Location","Num_Bathrooms","Num_Beds","BER_class","La
```

```
encoded_data = encoder.transform(df_numeric_test[columns_to_replace]) # Encode t
encoded_data = pd.DataFrame(encoded_data) # Convert this to a dataframe
encoded_data.columns = encoder.get_feature_names_out() # Rename encoded columns

for column in columns_to_replace: # Iterate through list of columns
    df_numeric_test.drop(column ,axis=1, inplace=True) # Drop (delete) the ones
df_numeric_test = pd.concat([encoded_data,df_numeric_test],axis=1) # Concatenate

scaler = MinMaxScaler() # Initialize scaler
for column in df_numeric_test.columns:
    df_numeric_test[column]=pd.DataFrame(scaler.fit_transform(df_numeric_test[co
X_numeric_test = df_numeric_test
y_pred = best_model.predict(X_numeric_test)
y_pred = scaler.inverse_transform(y_pred)
# save predictions in a file
df_id = pd.DataFrame(data=np.arange(1639,2341), columns = ['Index'])
df_class = pd.DataFrame(data=y_pred, columns = ['Price'])
df_pred = pd.concat([df_id, df_class], axis=1)

df_pred.to_csv('CS6501_Kaggle_TeamThree.csv', sep=',', index=False)
```

SUMMARY

Two techniques were used to successfully estimate house prices: the first, NLP, and the second, LSTM. We used one hot encoding to transform categorical data to continuous data/numerical data. The text in the Description, Services, and Features columns was filtered for stopwords and lemmatized to extract the word's root for NLP, following which the sentences were vectorized and a sparse matrix was created. The numerical and sparse matrices were combined. The information was then used to train various models. The xgboost model has the best accuracy, with an RME value of 13748.106.

REFERENCES

Raghavan, Shreyas. “Create a Model to Predict House Prices Using Python.” Medium, Towards Data Science, 20 June 2017, https://towardsdatascience.com/create-a-model-to-predict-house-prices-using-python-d34fe8fad88f.

mason, Britney. “House Price Prediction Using LSTM - Arxiv.” House Price Prediction, https://arxiv.org/pdf/1709.08432.

Mir, Mahsa. “House Prices Prediction Using Deep Learning.” Medium, Towards Data Science, 24 July 2020, https://towardsdatascience.com/house-prices-prediction-using-deep-learning-dea265cc3154.