CS985DLGL-GA2-BIN

April 11, 2021

1 CS985 Deep Learning Group L

1.1 Binary Classification Problem

Ian Richardson 202074007

Fraser Bayne 202053049

Slav Ivanov 201645797

Lora Kiosseva 202082329

In this document, binary classification of the provided dataset was run on 4 different models: - A Sci-Kit Learn standard model (One verses One) - A baseline 3 layer dense Neural Network - A deeper LSTM based neural net - The Wide and Deep model from the book

Each model runs its training separately then prints out a CSV (to the ./csv/ directory) of its predictions on the test dataset in the correct format for the Kaggle submission. The best performing models were the Baseline NN and the LSTM, producing the best Kaggle scores that we got of ~0.10

2 Method

After the data is read in from the CSV files provided we are dropping the columns from the dataset that we do not need such as "user", "session" and "query", as they do not hold information useful for predicting the relevancy of the document, only other semantic stuff.

After this, the columns which contain categorical data are one-hot encode, this way we change its type from strings to number data that can be used within the models and remove any relationships between these categories that should be unrelated.

The data is split into X and Y data for use in the models and a validation set of X and Y data split off for testing the accuracy of the models.

Random Over/Under samplers are used to augment the data to give us more to train from while keeping the proportions of the data the same.

The Wide and Deep model requires 2 X inputs, so copies of the data are created and the X data is cut vertically into 2 overlapping sections for use in this model.

2.1 Standard model - function: ovo()

The baseline standard model chosen for this was a One Verses One classifier from the Scikit-Learn library. This model was chosen for is performance on binary classification tasks.

2.2 Baseline NN - class: BaselineNN()

A baseline dense neural network, consisting of 3 hidden layers of 300 neurons each, each layers activation layer is ReLU, followed by the output layer of 1 sigmoid activated neuron.

2.3 LSTM Model - class: LSTMModel()

A much deeper LSTM network was created with 8 hidden layers, 3 LSTM layers with 3 dropout layers and a dense layer at the end, followed by the same 1 neuron output layer.

2.4 WideAndDeep Model: class: WideAndDeepModel()

The model WideAndDeep from the book "Hands on Machine Learning" [1], this model will take in 2 sets of X inputs to allow for more possible relationships to be modeled and was chosen due to the complex and multi-dimential nature of the data.

2.5 Training - function: trainModel()

This is a function that will take in a specified model and run the fit() function of that model to a set of data. This made running the models much easier as all we had to do was set up the data and pass it in as a parameter to this function to train the neural net type models. This function also will print out checkpoint files, useful for loading in the weights when you would not have time to run training for such a long time.

2.6 Predict - function: generateCSV

Finally, this function will take the trained model and get a prediction for the test dataset, writting it out to a CSV that can be submitted to Kaggle. The predictions were rounded as some models used sigmoid activations so all values between 0 and 1 could be written out, if the value is over 0.5 we would say that it predicted closer to a 1 so that value is rounded up. Each model writes out its raw predictions as a print before rounding to show what was predicted.

2.7 Testing

Through experimentation with the models' hyperparameters, different values produced different results. THe results gathered did not improve when changes were made, save from increaing the training times.

2.8 Performance

This configuration runs in relitivly quick time, with the slowest model being the LSTM taking ~20sec per epoch on a GPU. The OVO classifier would run considerably slower the more rows of data given, so only the first 2000 rows of training data was given to that model to complete its run.

[1]:
```python
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import numpy as np
import pandas as pd

import tensorflow as tf
from tensorflow import keras

from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler

from sklearn.svm import SVC
from sklearn.multiclass import OneVsOneClassifier

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, scale
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import f1_score, confusion_matrix

from skopt import gp_minimize, callbacks, load
from skopt.utils import cook_initial_point_generator
from skopt.space import Real

config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.compat.v1.InteractiveSession(config=config)
```

[2]:
```python
data = pd.read_csv('dataset/train.csv')
test_data = pd.read_csv('dataset/test.csv')
id_column = np.array(test_data["Id"]).reshape(-1,1)
```

```
c:\users\ian\appdata\local\programs\python\python38\lib\site-
packages\IPython\core\interactiveshell.py:3165: DtypeWarning: Columns (16) have
mixed types.Specify dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

[3]:
```python
# Columns we can drop
data = data.drop(columns=['user', 'session', 'query', 'timestamp',
                          'month', 'day', 'hour', 'cpvs'])

test_data = test_data.drop(columns=['user', 'session', 'query', 'timestamp',
                          'month', 'day', 'hour', 'cpvs', 'Id'])

data['nature'] = data['nature'].fillna('none')
test_data['nature'] = test_data['nature'].fillna('none')
```

[4]:
```python
def one_hot_encode(df, column):
    encoder = OneHotEncoder()
    genre_encoded, genre_categories = df[column].factorize()
    genre_1hot = encoder.fit_transform(genre_encoded.reshape(-1,1))
    enc_data = pd.DataFrame(genre_1hot.toarray())
    enc_data.columns = genre_categories
    enc_data.index = df.index
    df = df.join(enc_data)
    df = df.drop(columns=[column])
    return df

# Columns we want to one-hot encode
categorical_columns = ['search', 'source', 'type', 'nature']
for column in categorical_columns:
    data = one_hot_encode(data, column)
    test_data = one_hot_encode(test_data, column)
```

[5]:
```python
# What we want to predict
predict = 'psrel'

# Get everything except what we want to predict
X = np.array(data.drop([predict], 1)).astype(np.float64)
X_forreal = np.array(test_data).astype(np.float64)
# Column we want to predict
y = np.array(data[predict]).astype(np.float64)
```

[6]:
```python
print(X.shape, y.shape)
oversample = RandomOverSampler(sampling_strategy='auto')
undersample = RandomUnderSampler(sampling_strategy='auto')
X, y = oversample.fit_resample(X, y)
X, y = undersample.fit_resample(X, y)
print(X.shape, y.shape)
```

```
(33000, 30) (33000,)
(62076, 30) (62076,)
```

[7]:
```python
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.
 ↪2, random_state=42)
X_valid, X_train = X_train_full[:4000], X_train_full[4000:]
y_valid, y_train = y_train_full[:4000], y_train_full[4000:]

X_train_wide, X_train_deep = X_train[:,:22], X_train[:,7:]
X_valid_wide, X_valid_deep = X_valid[:,:22], X_valid[:,7:]
X_test_wide, X_test_deep = X_test[:,:22], X_test[:,7:]
```

[8]:
```python
def ovo():
    cutoff = 2000
    model = OneVsOneClassifier(SVC(kernel="poly", degree=2, C=1.1, gamma=0.3,
                                  probability=True, decision_function_shape =␣
↪"ovo",
                                  random_state=42))
    model.fit(scale(X_train_full[:cutoff, :]), y_train_full[:cutoff])
    y_pred = model.predict(scale(X_test[:cutoff, :]))
    print("> One Versus One Classifier", model.score(scale(X_test[:cutoff, :]),␣
↪y_test[:cutoff]),
          "- f1", f1_score(y_test[:cutoff], y_pred, average="weighted"))
    return model
```

[9]:
```
class BaselineNN(keras.models.Model):
    def __init__(self, units=300, activation="relu", **kwargs):
        super().__init__(**kwargs) # handles standard args (e.g., name)
        self.hidden1 = keras.layers.Dense(units, activation=activation)
        self.hidden2 = keras.layers.Dense(units, activation=activation)
        self.hidden3 = keras.layers.Dense(units, activation=activation)
        self.main_output = keras.layers.Dense(1, activation="sigmoid")

    def call(self, inputs):
        hidden1 = self.hidden1(inputs)
        hidden2 = self.hidden2(hidden1)
        hidden3 = self.hidden3(hidden2)
        main_output = self.main_output(hidden3)
        return main_output
```

[10]:
```
class LSTMModel(keras.models.Model):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.hidden1 = keras.layers.LSTM(256, return_sequences=True)
        self.hidden2 = keras.layers.Dropout(0.3)
        self.hidden3 = keras.layers.LSTM(512, return_sequences=True)
        self.hidden4 = keras.layers.Dropout(0.3)
        self.hidden5 = keras.layers.LSTM(256)
        self.hidden6 = keras.layers.Dense(256)
        self.hidden7 = keras.layers.Dropout(0.3)
        self.main_output = keras.layers.Dense(1, activation='sigmoid')

    def call(self, inputs):
        hidden1 = self.hidden1(inputs)
        hidden2 = self.hidden2(hidden1)
        hidden3 = self.hidden3(hidden2)
        hidden4 = self.hidden4(hidden3)
        hidden5 = self.hidden5(hidden4)
        hidden6 = self.hidden6(hidden5)
        hidden7 = self.hidden7(hidden6)
        main_output = self.main_output(hidden7)
        return main_output
```

[11]:
```
class WideAndDeepModel(keras.models.Model):
    def __init__(self, units=300, activation="relu", **kwargs):
        super().__init__(**kwargs)
        self.hidden1 = keras.layers.Dense(units, activation=activation)
        self.hidden2 = keras.layers.Dense(units, activation=activation)
        self.main_output = keras.layers.Dense(1, activation="sigmoid")

    def call(self, inputs):
        input_A, input_B = inputs
        hidden1 = self.hidden1(input_B)
        hidden2 = self.hidden2(hidden1)
        concat = keras.layers.concatenate([input_A, hidden2])
        main_output = self.main_output(concat)
        return main_output
```

[12]:
```
def loadWeights(model):
    weightsName = "L0.0473-B32-E3974.hdf5"
    print("loading weights", weightsName)
    model.load_weights(weightsName)
    return model
```

[13]:
```
def trainModel(model, xm_train, ym_train, xm_val, ym_val):
    opt = tf.keras.optimizers.Adam(learning_rate=0.001)
    model.compile(loss='binary_crossentropy', optimizer=opt,␣
 ↪metrics=["binary_accuracy"])
    b_size = 32
    eps = 5
    filepath = "weight-gen/L{loss:.4f}-B" + str(b_size) + "-E{epoch:02d}.hdf5"␣
 ↪# destination and name of saved checkpoint files
    checkpoint = keras.callbacks.ModelCheckpoint(filepath,
                                                 monitor="loss",
                                                 verbose=0,
                                                 save_best_only=True,
                                                 mode="min")
    model.fit(xm_train, ym_train, batch_size=b_size, epochs=eps,␣
 ↪callbacks=[checkpoint], validation_data=(xm_val, ym_val))
    y_pred = np.around(model.predict(xm_val))
    print("F1 SCORE: ", f1_score(ym_val, y_pred))
    return model
```

[14]:
```
def generateCSV(model, x, name):
    y_forreal = model.predict(x)
    print(y_forreal)
    y_forreal = np.around(y_forreal)

    if(name == "sklearnovo"):
        y_forreal = np.reshape(y_forreal, (-1, 1))
        print(y_forreal)

    csv = np.concatenate((id_column, y_forreal), axis=1).astype(np.int32)
    csv = np.vstack((np.array(["Id", predict]), csv))
    np.savetxt("./csv/" + name + ".csv", csv, fmt='%s', delimiter=",")
```

3 Results and Discussion

- **(Accuracy: 0.666 - f1: 0.664 - Kaggle: 0.07381)** - ovo()
- **(Accuracy: 0.724 - f1: 0.730 - Kaggle: 0.07625)** - BaselineNN()
- **(Accuracy: 0.774 - f1: 0.774 - Kaggle: 0.10526)** - LSTMModel()
- **(Accuracy: 0.650 - f1: 0.671 - Kaggle: 0.07334)** - WideAndDeepModel()

The baseline and LSTM models perform the best, both on Kaggle and on our validation set, the LSTM getting our highest score on Kaggle 0.105 and with longer training would perform even better. the OVO and WideAndDeep models do not perform as well, both converge fast and get lower scores on Kaggle and the more training we give it it does not improve.

The features selected to be most important to us are:

- search
- source
- type
- nature

The longer the model runs for and lower the loss value gets does not neccessarily correlate to Kaggle score performance, so sometimes running for many more epochs would not improve our scores and the best score generated was from the LSTM model with only 50 epochs

4 Summary and Recommendation

The system build can support the addition of more models as well as the models created in this are modular and could be used elsewhere in other systems for other problems. With the irrelivant columns dropped and categorical columns one-hot encoded, running each model with the data was much more straightforward and achieved better results.

5 References

1. Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, Aurélien Géron **(For the WideAndDeep Model)**

6 Training and Validating

6.1 Training of the standard ML model - ovo()

- This only trains on the first 2000 rows of training data

[15]:
```
modelSklearn = ovo()
```

```
> One Versus One Classifier 0.666 - f1 0.6640419615285641
```

6.2 Training of the baseline NN model - BaselineNN()

[16]:
```
modelBase = trainModel(BaselineNN(), X_train, y_train, X_valid, y_valid)
```

```
Epoch 1/5
1427/1427 [==============================] - 12s 4ms/step - loss: 0.8752 -
binary_accuracy: 0.6260 - val_loss: 0.5742 - val_binary_accuracy: 0.6743
Epoch 2/5
1427/1427 [==============================] - 5s 3ms/step - loss: 0.5656 -
binary_accuracy: 0.6917 - val_loss: 0.5415 - val_binary_accuracy: 0.7007
Epoch 3/5
1427/1427 [==============================] - 5s 3ms/step - loss: 0.5382 -
binary_accuracy: 0.7082 - val_loss: 0.5197 - val_binary_accuracy: 0.7235
Epoch 4/5
1427/1427 [==============================] - 5s 3ms/step - loss: 0.5083 -
binary_accuracy: 0.7305 - val_loss: 0.5091 - val_binary_accuracy: 0.7303
Epoch 5/5
1427/1427 [==============================] - 5s 3ms/step - loss: 0.4919 -
binary_accuracy: 0.7412 - val_loss: 0.5062 - val_binary_accuracy: 0.7240
F1 SCORE:  0.7309941520467835
```

6.3 Training of the LSTM Model - LSTMModel()

- Data needs to first be reshaped to be used by this model

[17]:
```
lstm_in = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
lstm_val = np.reshape(X_valid, (X_valid.shape[0], X_valid.shape[1], 1))
modelLSTM = trainModel(LSTMModel(), lstm_in, y_train, lstm_val, y_valid)
```

```
Epoch 1/5
1427/1427 [==============================] - 39s 19ms/step - loss: 0.6837 -
binary_accuracy: 0.5530 - val_loss: 0.6563 - val_binary_accuracy: 0.6047
Epoch 2/5
1427/1427 [==============================] - 26s 18ms/step - loss: 0.6312 -
binary_accuracy: 0.6252 - val_loss: 0.5469 - val_binary_accuracy: 0.7237
Epoch 3/5
1427/1427 [==============================] - 26s 18ms/step - loss: 0.5512 -
binary_accuracy: 0.7270 - val_loss: 0.5307 - val_binary_accuracy: 0.7333
Epoch 4/5
1427/1427 [==============================] - 26s 18ms/step - loss: 0.5276 -
binary_accuracy: 0.7403 - val_loss: 0.5110 - val_binary_accuracy: 0.7473
Epoch 5/5
1427/1427 [==============================] - 26s 18ms/step - loss: 0.5125 -
binary_accuracy: 0.7499 - val_loss: 0.5044 - val_binary_accuracy: 0.7598
F1 SCORE:  0.7736160188457009
```

6.4 Training of the WideAndDeep Model - WideAndDeepModel()

- This model will require 2 X inputs, seporated earlier

[18]:
```
modelWAD = trainModel(WideAndDeepModel(), [X_test_wide, X_test_deep], y_test,␣
↪[X_valid_wide, X_valid_deep], y_valid)
```

```
Epoch 1/5
388/388 [==============================] - 2s 3ms/step - loss: 0.7546 -
binary_accuracy: 0.6160 - val_loss: 0.6355 - val_binary_accuracy: 0.6538
Epoch 2/5
388/388 [==============================] - 1s 3ms/step - loss: 0.6365 -
binary_accuracy: 0.6459 - val_loss: 0.6420 - val_binary_accuracy: 0.6455
Epoch 3/5
388/388 [==============================] - 1s 3ms/step - loss: 0.6382 -
binary_accuracy: 0.6518 - val_loss: 0.6443 - val_binary_accuracy: 0.6357
Epoch 4/5
388/388 [==============================] - 1s 3ms/step - loss: 0.6317 -
binary_accuracy: 0.6532 - val_loss: 0.6311 - val_binary_accuracy: 0.6568
Epoch 5/5
388/388 [==============================] - 1s 3ms/step - loss: 0.6283 -
binary_accuracy: 0.6578 - val_loss: 0.6294 - val_binary_accuracy: 0.6497
F1 SCORE:  0.670585469080649
```

7 Predictions

- Each model is passed into the generateCSV() function to get its prediction on the test data set and write it out to a CSV file for submission on Kaggle

[19]:
```
generateCSV(modelSklearn, X_forreal, "sklearnovo")
```

```
[0. 1. 1. … 1. 1. 1.]
[[0.]
 [1.]
 [1.]
 …
 [1.]
 [1.]
 [1.]]
```

[20]:
```
generateCSV(modelBase, X_forreal, "base")
```

```
[[0.42983472]
 [0.2509963 ]
 [0.66495335]
 …
 [0.42983472]
 [0.59225297]
 [0.6720405 ]]
```

[21]:
```
lstm_forreal = np.reshape(X_forreal, (X_forreal.shape[0], X_forreal.shape[1],␣
↪1))
generateCSV(modelLSTM, lstm_forreal, "lstm")
```

```
[[0.52756566]
 [0.2173662 ]
 [0.3529291 ]
 …
 [0.71345186]
 [0.4166437 ]
 [0.3464281 ]]
```

[22]:
```
X_forreal_wide, X_forreal_deep = X_forreal[:,:22], X_forreal[:,7:]
generateCSV(modelWAD, [X_forreal_wide, X_forreal_deep], "wad")
```

```
[[0.3242547 ]
 [0.32384267]
 [0.30399007]
 …
 [0.02773135]
 [0.53214264]
 [0.50087696]]
```