

MODULE 2

ICP-5

In class programming:

1. Save the model and use the saved model to predict on new text data (ex, “A lot of good things are happening. We are respected again throughout the world, and that's a great thing.”)

The screenshot displays a Google Colab notebook titled 'MICP5.ipynb'. The notebook is open to a cell where a Keras model is saved and then used to predict the sentiment of a new text input. The code in the cell is as follows:

```
[79] model.save("model.h5")

EXTRACTING MODEL

[88] from keras.models import load_model
model_save = load_model('content/model.h5')

/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning: Converting sparse IndexedSlices to
"Converting sparse IndexedSlices to a dense Tensor of unknown shape."

PREDICTING GIVEN RESULTS

[81] import numpy as np
sentence = ['A lot of good things are happening. We are respected again throughout the world, and that is a great thing']
#vectorizing the tweet by the pre-fitted tokenizer instance
sentence = tokenizer.texts_to_sequences(sentence)
#padding the tweet to have exactly the same shape as 'embedding_2' input
sentence = pad_sequences(sentence, maxlen=28, dtype='int32', value=0)

sentiment = model_save.predict_classes(sentence, batch_size=1)[0]

if sentiment == 0:
    print("negative")
elif sentiment == 1:
    print("neutral")
else:
    print("positive")

positive
```

The notebook interface shows a file explorer on the left with a folder named 'sample_data' containing 'Sentiment.csv', 'model.h5', and 'spam.csv'. The right sidebar shows a table of data from 'spam.csv' with columns 'v1' and 'v2'. The table contains 100 entries, with the first few rows showing examples of spam and ham messages.

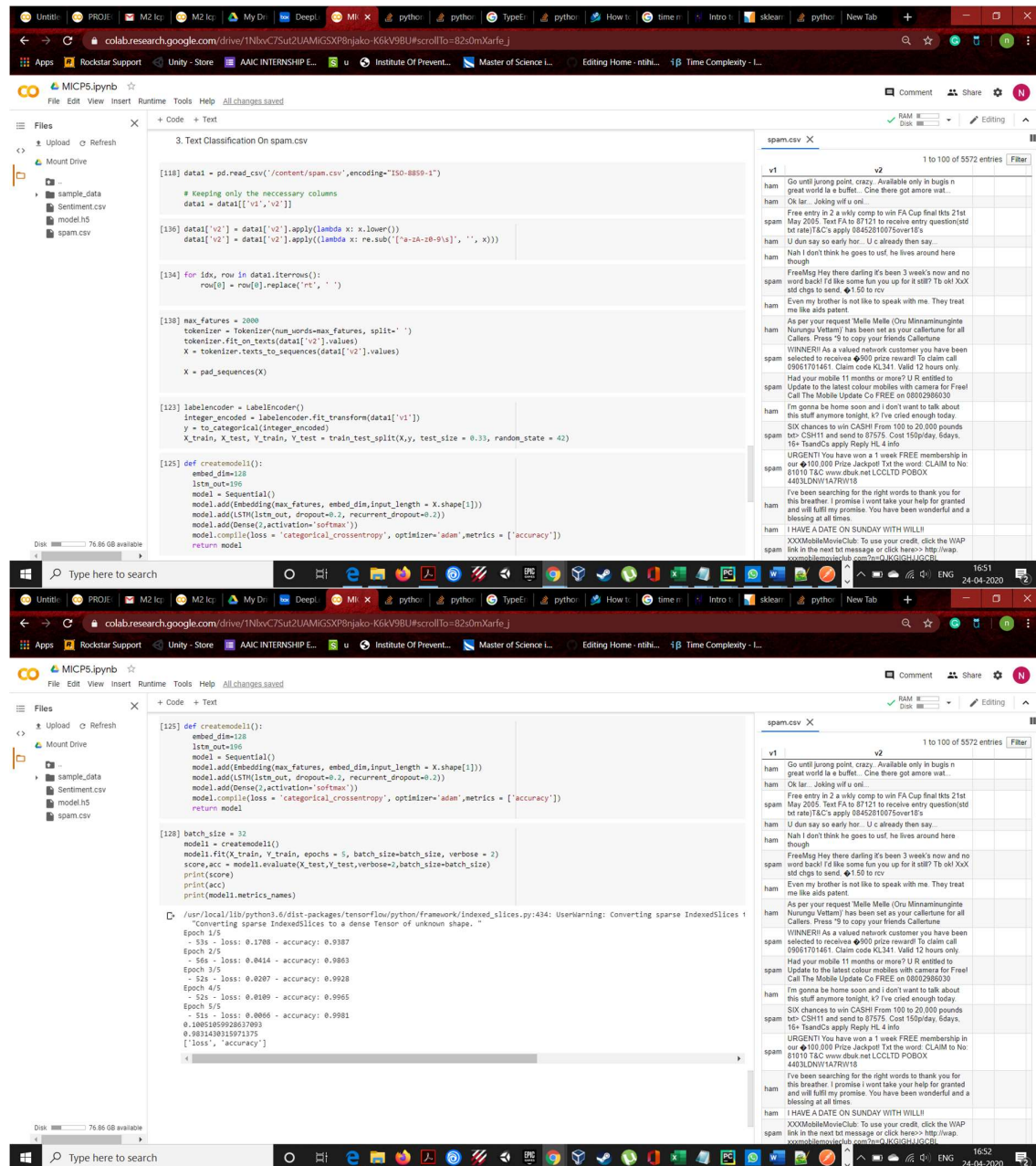
2. Apply GridSearchCV on the source code provided in the class

The image displays two screenshots of a Google Colab notebook, illustrating the application of GridSearchCV to a Keras model.

Top Screenshot: The notebook shows the initial code for training a Keras model. The code defines a KerasClassifier, sets hyperparameters (batch_size, epochs, param_grid), and uses GridSearchCV to find the best model. The output shows the first 10 epochs of training results, including loss and accuracy for each epoch.

Bottom Screenshot: The notebook shows the code after modification to use GridSearchCV. The code defines a KerasClassifier, sets hyperparameters (batch_size, epochs, param_grid), and uses GridSearchCV to find the best model. The output shows the first 10 epochs of training results for the best model found, including loss and accuracy for each epoch.

3. Apply the code on spam data set available in the source code (text classification on the spam. csv data set)



```
[118] data = pd.read_csv('/content/spam.csv', encoding='ISO-8859-1')
# keeping only the necessary columns
data = data[['v1', 'v2']]

[136] data['v2'] = data['v2'].apply(lambda x: x.lower())
data['v2'] = data['v2'].apply(lambda x: re.sub('[^a-zA-Z0-9\s]', '', x))

[134] for idx, row in data.iterrows():
    row['v1'] = row['v1'].replace('rt', ' ')

[138] max_features = 20000
tokenizer = Tokenizer(num_words=max_features, split=' ')
tokenizer.fit_on_texts(data['v2'].values)
X = tokenizer.texts_to_sequences(data['v2'].values)
X = pad_sequences(X)

[123] labelencoder = LabelEncoder()
integer_encoded = labelencoder.fit_transform(data['v1'])
y = to_categorical(integer_encoded)
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = 0.33, random_state = 42)

[125] def createmodel():
    embed_dim=128
    ltrn_out=106
    model = Sequential()
    model.add(Embedding(max_features, embed_dim, input_length = x.shape[1]))
    model.add(LSTM(ltrn_out, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(1, activation='softmax'))
    model.compile(loss = 'categorical_crossentropy', optimizer='adam', metrics = ['accuracy'])
    return model

[128] batch_size = 32
model1 = createmodel()
model1.fit(X_train, Y_train, epochs = 5, batch_size=batch_size, verbose = 2)
score, acc = model1.evaluate(X_test, Y_test, verbose=2, batch_size=batch_size)
print(score)
print(acc)
print(model1.metrics_names)
```

Converting sparse IndexedSlices to a dense Tensor of unknown shape.

Epoch 1/5
- 53s - loss: 0.1788 - accuracy: 0.9387

Epoch 2/5
- 56s - loss: 0.0414 - accuracy: 0.9863

Epoch 3/5
- 52s - loss: 0.0207 - accuracy: 0.9928

Epoch 4/5
- 52s - loss: 0.0189 - accuracy: 0.9965

Epoch 5/5
- 51s - loss: 0.0066 - accuracy: 0.9981

0.1085185922673903
0.983140815971275
['loss', 'accuracy']

By,

DUKKIPATI SRI SAI NITHIN CHOWDARY

CLASS ID: 4