

```

{
  "cells": [
    {
      "cell_type": "markdown",
      "metadata": {},
      "source": [
        "## Assignment 4 Solution"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {},
      "source": [
        "### Import necessary libraries"
      ]
    },
    {
      "cell_type": "code",
      "execution_count": 2,
      "metadata": {},
      "outputs": [],
      "source": [
        "import pandas as pd\n",
        "import numpy as np\n",
        "from sklearn.model_selection import train_test_split\n",
        "from sklearn.preprocessing import LabelEncoder\n",
        "from tensorflow.keras.models import Sequential\n",
        "from tensorflow.keras.layers import LSTM, Dense, Dropout, Embedding\n",
        "from tensorflow.keras.optimizers import RMSprop\n",
        "from tensorflow.keras.preprocessing.text import Tokenizer\n",
        "from tensorflow.keras.preprocessing import sequence"
      ]
    },
    {
      "cell_type": "markdown",
      "metadata": {},
      "source": [
        "### Data Pre-Processing"
      ]
    },
    {
      "cell_type": "code",
      "execution_count": 3,
      "metadata": {},
      "outputs": [
        {
          "data": {
            "text/html": [
              "<div>\n",
              "<style scoped>\n",
              "    .dataframe tbody tr th:only-of-type {\n",
              "        vertical-align: middle;\n",
              "    }\n",
              "\n",
              "    .dataframe tbody tr th {\n",
              "        vertical-align: top;\n",
              "    }\n",
            ]
          }
        }
      ]
    }
  ]
}

```

```

"\n",
"    .dataframe thead th {\n",
"        text-align: right;\n",
"    }\n",
"</style>\n",
"<table border=\"1\" class=\"dataframe\">\n",
"    <thead>\n",
"        <tr style=\"text-align: right;\">\n",
"            <th></th>\n",
"            <th>v1</th>\n",
"            <th>v2</th>\n",
"            <th>Unnamed: 2</th>\n",
"            <th>Unnamed: 3</th>\n",
"            <th>Unnamed: 4</th>\n",
"        </tr>\n",
"    </thead>\n",
"    <tbody>\n",
"        <tr>\n",
"            <th>0</th>\n",
"            <td>ham</td>\n",
"            <td>Go until jurong point, crazy.. Available only ...</td>\n",
"            <td>NaN</td>\n",
"            <td>NaN</td>\n",
"            <td>NaN</td>\n",
"        </tr>\n",
"        <tr>\n",
"            <th>1</th>\n",
"            <td>ham</td>\n",
"            <td>Ok lar... Joking wif u oni...</td>\n",
"            <td>NaN</td>\n",
"            <td>NaN</td>\n",
"            <td>NaN</td>\n",
"        </tr>\n",
"        <tr>\n",
"            <th>2</th>\n",
"            <td>spam</td>\n",
"            <td>Free entry in 2 a wkly comp to win FA Cup fina...</td>\n",
"            <td>NaN</td>\n",
"            <td>NaN</td>\n",
"            <td>NaN</td>\n",
"        </tr>\n",
"        <tr>\n",
"            <th>3</th>\n",
"            <td>ham</td>\n",
"            <td>U dun say so early hor... U c already then say...</td>\n",
"            <td>NaN</td>\n",
"            <td>NaN</td>\n",
"            <td>NaN</td>\n",
"        </tr>\n",
"        <tr>\n",
"            <th>4</th>\n",
"            <td>ham</td>\n",
"            <td>Nah I don't think he goes to usf, he lives aro...</td>\n",
"            <td>NaN</td>\n",
"            <td>NaN</td>\n",
"            <td>NaN</td>\n",
"        </tr>\n",

```

```

" </tbody>\n",
"</table>\n",
"</div>"
],
"text/plain": [
"      v1                                v2 Unnamed: 2
\\n",
"0    ham  Go until jurong point, crazy.. Available only ...      NaN
\n",
"1    ham                                Ok lar... Joking wif u oni...      NaN
\n",
"2    spam  Free entry in 2 a wkly comp to win FA Cup fina...      NaN
\n",
"3    ham  U dun say so early hor... U c already then say...      NaN
\n",
"4    ham  Nah I don't think he goes to usf, he lives aro...      NaN
\n",
"\n",
"  Unnamed: 3 Unnamed: 4  \n",
"0      NaN      NaN  \n",
"1      NaN      NaN  \n",
"2      NaN      NaN  \n",
"3      NaN      NaN  \n",
"4      NaN      NaN  "
]
},
"execution_count": 3,
"metadata": {},
"output_type": "execute_result"
}
],
"source": [
"df = pd.read_csv('spam.csv', delimiter=',', encoding='latin-1')\n",
"df.head()"
]
},
{
"cell_type": "code",
"execution_count": 4,
"metadata": {},
"outputs": [
{
"name": "stdout",
"output_type": "stream",
"text": [
"<class 'pandas.core.frame.DataFrame'>\n",
"RangeIndex: 5572 entries, 0 to 5571\n",
"Data columns (total 2 columns):\n",
"#      Column  Non-Null Count  Dtype \n",
"---  -
"0      v1      5572 non-null  object\n",
"1      v2      5572 non-null  object\n",
"dtypes: object(2)\n",
"memory usage: 87.2+ KB\n"
]
}
]
}
],

```

```

    "source": [
        "df.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis=1,
inplace=True)\n",
        "df.info()"
    ]
},
{
    "cell_type": "code",
    "execution_count": 5,
    "metadata": {},
    "outputs": [],
    "source": [
        "X = df.v2\n",
        "Y = df.v1\n",
        "encoder = LabelEncoder()\n",
        "Y = encoder.fit_transform(Y)\n",
        "Y = Y.reshape(-1,1)"
    ]
},
{
    "cell_type": "code",
    "execution_count": 6,
    "metadata": {},
    "outputs": [],
    "source": [
        "X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2)"
    ]
},
{
    "cell_type": "code",
    "execution_count": 7,
    "metadata": {},
    "outputs": [],
    "source": [
        "tokenizer = Tokenizer(num_words=2000, lower=True)\n",
        "tokenizer.fit_on_texts(X_train)\n",
        "sequences = tokenizer.texts_to_sequences(X_train)\n",
        "X_train = sequence.pad_sequences(sequences, maxlen=200)"
    ]
},
{
    "cell_type": "markdown",
    "metadata": {},
    "source": [
        "### Create model"
    ]
},
{
    "cell_type": "code",
    "execution_count": 8,
    "metadata": {},
    "outputs": [],
    "source": [
        "model = Sequential()"
    ]
},

```

```

{
  "cell_type": "markdown",
  "metadata": {},
  "source": [
    "### Add layers"
  ]
},
{
  "cell_type": "code",
  "execution_count": 9,
  "metadata": {},
  "outputs": [],
  "source": [
    "model.add(Embedding(2000, 50, input_length=200))\n",
    "model.add(LSTM(64))\n",
    "model.add(Dense(256, activation=\"relu\"))\n",
    "model.add(Dropout(0.5))\n",
    "model.add(Dense(1, activation=\"sigmoid\"))"
  ]
},
{
  "cell_type": "code",
  "execution_count": 10,
  "metadata": {},
  "outputs": [
    {
      "name": "stdout",
      "output_type": "stream",
      "text": [
        "Model: \"sequential\"\n",
        "
        | Layer (type)                | Output Shape         | Param # |
        |=====|
        | embedding (Embedding)       | (None, 200, 50)     | 100000  |
        | lstm (LSTM)                  | (None, 64)           | 29440   |
        | dense (Dense)                | (None, 256)          | 16640   |
        | dropout (Dropout)           | (None, 256)          | 0       |
        | dense_1 (Dense)             | (None, 1)            | 257     |
        |=====|
        | Total params: 146,337\n",
        | Trainable params: 146,337\n",
        | Non-trainable params: 0\n",
        |=====|
      ]
    }
  ]
},
{
  "source": [
    "model.summary()"
  ]
},
{
  "cell_type": "markdown",

```

```

    "metadata": {},
    "source": [
        "### Compile the model"
    ]
},
{
    "cell_type": "code",
    "execution_count": 11,
    "metadata": {},
    "outputs": [],
    "source": [
        "model.compile(loss='binary_crossentropy', optimizer=RMSprop(),
metrics=['accuracy'])"
    ]
},
{
    "cell_type": "markdown",
    "metadata": {},
    "source": [
        "### Fit the model"
    ]
},
{
    "cell_type": "code",
    "execution_count": 12,
    "metadata": {},
    "outputs": [
        {
            "name": "stdout",
            "output_type": "stream",
            "text": [
                "Epoch 1/10\n",
                "28/28 [=====] - 4s 49ms/step - loss: 0.3426 - accuracy: 0.8738 - val_loss: 0.1774 - val_accuracy: 0.9585\n",
                "Epoch 2/10\n",
                "28/28 [=====] - 1s 27ms/step - loss: 0.1026 - accuracy: 0.9745 - val_loss: 0.0607 - val_accuracy: 0.9809\n",
                "Epoch 3/10\n",
                "28/28 [=====] - 1s 27ms/step - loss: 0.0417 - accuracy: 0.9882 - val_loss: 0.0606 - val_accuracy: 0.9832\n",
                "Epoch 4/10\n",
                "28/28 [=====] - 1s 27ms/step - loss: 0.0253 - accuracy: 0.9927 - val_loss: 0.0579 - val_accuracy: 0.9843\n",
                "Epoch 5/10\n",
                "28/28 [=====] - 1s 26ms/step - loss: 0.0191 - accuracy: 0.9947 - val_loss: 0.0744 - val_accuracy: 0.9865\n",
                "Epoch 6/10\n",
                "28/28 [=====] - 1s 28ms/step - loss: 0.0131 - accuracy: 0.9961 - val_loss: 0.0762 - val_accuracy: 0.9865\n",
                "Epoch 7/10\n",
                "28/28 [=====] - 1s 26ms/step - loss: 0.0085 - accuracy: 0.9969 - val_loss: 0.1080 - val_accuracy: 0.9854\n",
                "Epoch 8/10\n",
                "28/28 [=====] - 1s 26ms/step - loss: 0.0075 - accuracy: 0.9978 - val_loss: 0.0998 - val_accuracy: 0.9809\n",
                "Epoch 9/10\n",

```

```

        "28/28 [=====] - 1s 26ms/step - loss: 0.0053 -
accuracy: 0.9978 - val_loss: 0.1187 - val_accuracy: 0.9843\n",
        "Epoch 10/10\n",
        "28/28 [=====] - 1s 26ms/step - loss: 0.0031 -
accuracy: 0.9994 - val_loss: 0.1409 - val_accuracy: 0.9843\n"
    ]
},
{
    "data": {
        "text/plain": [
            "<keras.callbacks.History at 0x1ecbb7a4ee0>"
        ]
    },
    "execution_count": 12,
    "metadata": {},
    "output_type": "execute_result"
}
],
"source": [
    "model.fit(X_train, y_train, batch_size=128, epochs=10,
validation_split=0.2)"
]
},
{
    "cell_type": "markdown",
    "metadata": {},
    "source": [
        "### Save the model"
    ]
},
{
    "cell_type": "code",
    "execution_count": 13,
    "metadata": {},
    "outputs": [],
    "source": [
        "model.save(\"model.h5\")"
    ]
},
{
    "cell_type": "markdown",
    "metadata": {},
    "source": [
        "### Test the model"
    ]
},
{
    "cell_type": "code",
    "execution_count": 14,
    "metadata": {},
    "outputs": [],
    "source": [
        "test_sequences = tokenizer.texts_to_sequences(X_test)\n",
        "X_test = sequence.pad_sequences(test_sequences, maxlen=200)"
    ]
},
{

```

```

"cell_type": "code",
"execution_count": 15,
"metadata": {},
"outputs": [
  {
    "name": "stdout",
    "output_type": "stream",
    "text": [
      "35/35 [=====] - 0s 12ms/step - loss: 0.0825 -
accuracy: 0.9839\n"
    ]
  }
],
"source": [
  "acc = model.evaluate(X_test, y_test)"
]
},
{
  "cell_type": "code",
  "execution_count": 16,
  "metadata": {},
  "outputs": [],
  "source": [
    "def predict(message):\n",
    "    txt = tokenizer.texts_to_sequences(message)\n",
    "    txt = sequence.pad_sequences(txt, maxlen=200)\n",
    "    preds = model.predict(txt)\n",
    "    if preds > 0.5:\n",
    "        print(\"Spam\")\n",
    "    else:\n",
    "        print(\"Not Spam\")"
  ]
},
{
  "cell_type": "code",
  "execution_count": 19,
  "metadata": {},
  "outputs": [
    {
      "name": "stdout",
      "output_type": "stream",
      "text": [
        "1/1 [=====] - 0s 28ms/step\n",
        "Not Spam\n"
      ]
    }
  ]
},
"source": [
  "predict([\"Sorry, I'll call after the meeting.\"])"
]
},
{
  "cell_type": "code",
  "execution_count": 20,
  "metadata": {},
  "outputs": [
    {

```



```

    "name": "stdout",
    "output_type": "stream",
    "text": [
      "1/1 [=====] - 0s 25ms/step\n",
      "Spam\n"
    ]
  },
  ],
  "source": [
    "predict([\\"Congratulations!!! You won $50,000. Send message LUCKY100 to\nXXXXXXXXXX to recieve your prize.\"])"
  ]
},
],
"metadata": {
  "kernelspec": {
    "display_name": "Python 3.10.7 ('venv': venv)",
    "language": "python",
    "name": "python3"
  },
  "language_info": {
    "codemirror_mode": {
      "name": "ipython",
      "version": 3
    },
    "file_extension": ".py",
    "mimetype": "text/x-python",
    "name": "python",
    "nbconvert_exporter": "python",
    "pygments_lexer": "ipython3",
    "version": "3.10.7"
  },
  "orig_nbformat": 4,
  "vscode": {
    "interpreter": {
      "hash":
"642c4c79e4b143d0da89805c64dad1e5ef21fdea6c19e5430701ee75c9149d19"
    }
  },
  "nbformat": 4,
  "nbformat_minor": 2
}

```