## گزارش پروژه

در این پروژه قصد داریم تا داده‌های مربوط به نظرات خریداران ساعت‌های سایت آمازون را مورد تحلیل قرار دهیم و با مدل‌های Classification این نظرات را دسته‌بندی کنیم. برای اینکار ابتدا لازم است تا دیتاست را وارد کنیم. از دستور !wget در محیط Colab استفاده می‌کنیم.

```
[ ]  !wget "https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Watches_v1_00.tsv.gz"

     --2023-07-02 04:23:43--  https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Watches_v1_00.tsv.
     Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.216.249.214, 52.217.174.248, 52.216.170.141, ...
     Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.216.249.214|:443... connected.
     HTTP request sent, awaiting response... 200 OK
     Length: 162973819 (155M) [application/x-gzip]
     Saving to: ‘amazon_reviews_us_Watches_v1_00.tsv.gz’

     amazon_reviews_us_W 100%[===================>] 155.42M  86.3MB/s    in 1.8s

     2023-07-02 04:23:45 (86.3 MB/s) - ‘amazon_reviews_us_Watches_v1_00.tsv.gz’ saved [162973819/162973819]
```

حال لازم است از فایل زیپ شده استخراج کنیم. برای اینکار از دستور !gzip -d در محیط colab استفاده می‌کنیم.

```
[ ]  !gzip -d "/content/amazon_reviews_us_Watches_v1_00.tsv.gz"
```

پس از استخراج سازی، می‌توانیم فایل با پسوند .tsv را با دستور pd.read_csv که در کتابخانه pandas موجود است، را بارگذاری کنیم. لازم به ذکر است به دلیل حجیم بودن داده‌ها تنها ۱۰۰ هزار سمپل از داده‌ها استفاده می‌شود تا زمان آموزش مدل و شبکه بسیار طولانی نگردد. همچنین ردیف‌هایی که دارای مشکلاتی از بابت داده هستند با دستور error_bad_lines=False رد می‌کنیم.

```
import pandas as pd
file_path = '/content/amazon_reviews_us_Watches_v1_00.tsv'
df = pd.read_csv(file_path, sep='\t',error_bad_lines=False)
Data_used = 100000
df = df.iloc[:Data_used,:]
df.shape
```

```
<ipython-input-3-30223d02b6fb>:3: FutureWarning: The error_bad_lines argument has been deprecated and wi


  df = pd.read_csv(file_path, sep='\t',error_bad_lines=False)
Skipping line 8704: expected 15 fields, saw 22
Skipping line 16933: expected 15 fields, saw 22
Skipping line 23726: expected 15 fields, saw 22

Skipping line 85637: expected 15 fields, saw 22

Skipping line 132136: expected 15 fields, saw 22
Skipping line 158070: expected 15 fields, saw 22
Skipping line 166007: expected 15 fields, saw 22
Skipping line 171877: expected 15 fields, saw 22
Skipping line 177756: expected 15 fields, saw 22
Skipping line 181773: expected 15 fields, saw 22
Skipping line 191085: expected 15 fields, saw 22
Skipping line 196273: expected 15 fields, saw 22
Skipping line 196331: expected 15 fields, saw 22
```

پس از اینکار لازم است تا ردیف‌هایی که دارای داده NaN هستند را از دیتاست حذف کنیم. برای اینکار از دستور dropna() در کتابخانه pandas استفاده می‌کنیم.

```
# Drop null values and count rows
df = df.dropna()
num_rows = df.count()
num_rows
```

```
marketplace          99935
customer_id          99935
review_id            99935
product_id           99935
product_parent       99935
product_title        99935
product_category     99935
star_rating          99935
helpful_votes        99935
total_votes          99935
vine                 99935
verified_purchase    99935
review_headline      99935
review_body          99935
review_date          99935
dtype: int64
```

## گزارش پروژه

این دیتاست دارای ستون star_rating است که از شماره ۱ تا ۵ امتیازدهی شده‌اند توسط کاربران. این امتیازدهی بصورت جدول زیر است. به علت آنکه در صورت پروژه گفته شده است باید تحلیل بصورت دو کلاسه انجام شود لذا از امتیازهای ۵ و ۱ استفاده می‌کنیم.

```
review=pd.DataFrame(df.groupby('star_rating').size().sort_values(ascending=False).rename('No of Users').reset_index())
review.head()
```

| | star_rating | No of Users |
|---|---|---|
| 0 | 5 | 61116 |
| 1 | 4 | 16199 |
| 2 | 1 | 9585 |
| 3 | 3 | 7748 |
| 4 | 2 | 5287 |

بر این اساس دیتافریمی تشکیل می‌دهیم که شامل ستون‌های امتیازها و نظرات کاربران است.

```
permanent = df[['star_rating' , 'review_body']]
mpermanent=permanent.dropna()
mpermanent.head()
```

| | star_rating | review_body |
|---|---|---|
| 21667 | 5 | Great watch, very classy. Really liking it. |
| 21652 | 2 | Pros: Face of the watch was attractive.<br />C... |
| 444543 | 5 | Wife loves it, just what she wanted, unfortuna... |
| 91226 | 5 | I absolutely love this watch. I had my eyes o... |
| 734009 | 5 | Llego en perfecto estado, estoy muy contenta y... |

پس از آن تنها ردیف‌هایی انتخاب می‌شوند که دارای امتیازهای ۱ و ۵ هستند. خروجی بصورت زیر خواهد بود:

```
actualrating = mpermanent[(mpermanent['star_rating'] == 1) | (mpermanent['star_rating'] == 5)]
actualrating.shape
```

```
(68342, 2)
```

در نهایت ورودی‌ها و خروجی‌ها را جدا می‌کنیم:

```
y = actualrating['star_rating']
x = actualrating['review_body'].reset_index()
print(len(y))
X = x['review_body']
print(X)
```

```
68342
0              Great watch, very classy. Really liking it.
1        Wife loves it, just what she wanted, unfortuna...
2        I absolutely love this watch.  I had my eyes o...
3        Llego en perfecto estado, estoy muy contenta y...
4        Seiko makes a great watch, period. I don't buy...
                               ...
68337    What a beautiful watch. I purchased this for m...
68338    elegant...such nice looking watch!!! my dad lo...
68339    Run for the hills. Don't waste your money and ...
68340    Looks great!  If you want a watch for every da...
68341    This watch is better than I expected.....you c...
Name: review_body, Length: 68342, dtype: object
```

حال لازم است تابعی بنویسیم که همه فرایندهای Preprocessing ها را شامل شود. این فرایندها شامل RegexpTokenizer، Tokenization، Stop words، Lemmatization و Stemming می‌باشد. این تابع بصورت زیر می‌باشد:

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer, PorterStemmer
from nltk.tokenize import word_tokenize
from nltk.tokenize import RegexpTokenizer

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
tokenizer = RegexpTokenizer(r'\w+')

lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))
stemmer = PorterStemmer()


def preprocess_text(text):
    tokenized =  ' '.join(tokenizer.tokenize(text))
    tokenized = tokenized.replace('_', ' ')
    # Tokenization
    tokens = word_tokenize(tokenized.lower())
    # Stop words removal
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # Lemmatization
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in filtered_tokens]
    # Stemming
    stemmed_tokens = [stemmer.stem(token) for token in lemmatized_tokens]
    return stemmed_tokens
```

**گزارش پروژه**

## قسمت الف)

حال از تابع CountVectorizer از کتابخانه Sklearn استفاده می‌کنیم تا داده‌های متنی را به ماتریس‌های ویژگی تبدیل کنیم. تابع ورودی این تابع را همان تابع Preprocess_text قرار می‌دهیم:

```
from sklearn.feature_extraction.text import CountVectorizer

bow_transformer = CountVectorizer(analyzer=preprocess_text).fit(X)
```

سپس مدلی که یادگیری برایش انجام شد را بر روی داده‌های ورودی پیاده‌سازی میکنیم و از متد transform استفاده می‌کنیم:

```
[ ]  X = bow_transformer.transform(X)
```

حال لازم است تا داده‌های خام X و Y را به دو بخش آموزش و تست تقسیم کنیم. برای آنکه در مدل‌های یادگیری ماشین داده Validation نداریم لذا تعداد داده تست را برابر با ۳۰ درصد درنظر می‌گیریم.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

حال از مدل MultinomialNB که مدل Naïve bayes می‌باشد استفاده می‌کنیم. ابتدا بر روی داده‌های آموزش آموزش را انجام می‌دهیم و سپس برای داده‌های تست فرایند پیش‌بینی را انجام می‌دهیم.

```
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()
nb.fit(X_train, y_train)
nb_preds = nb.predict(X_test)
```

حال گزارش Classification و ماتریس پیچیدگی آن را بدست می‌آوریم. برای اینکار از توابع Classification report و confusion_matrix که در کتابخانه sklearn هستند استفاده می‌کنیم.

```
from sklearn.metrics import confusion_matrix, classification_report
print(confusion_matrix(y_test, nb_preds))
print('\n')
print(classification_report(y_test, nb_preds))
nb.score(X_train, y_train)
```

```
[[ 1789   823]
 [  441 17450]]


              precision    recall  f1-score   support

           1       0.80      0.68      0.74      2612
           5       0.95      0.98      0.97     17891

    accuracy                           0.94     20503
   macro avg       0.88      0.83      0.85     20503
weighted avg       0.94      0.94      0.94     20503

0.9510441271765714
```

حال مشابه با همین کار از مدل LogisticRegression در کتابخانه Sklearn استفاده می‌کنیم و خروجی‌های مشابه دریافت می‌کنیم.

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state=0)
clf.fit(X, y)
clf_preds = clf.predict(X_test)

print(confusion_matrix(y_test, clf_preds))
print('\n')
print(classification_report(y_test, clf_preds))
nb.score(X_train, y_train)
```

```
[[ 2170   442]
 [  124 17767]]


              precision    recall  f1-score   support

           1       0.95      0.83      0.88      2612
           5       0.98      0.99      0.98     17891

    accuracy                           0.97     20503
   macro avg       0.96      0.91      0.93     20503
weighted avg       0.97      0.97      0.97     20503
```

همانطور که مشاهده می‌شود دقت خروجی مدل Logistic از مدل Naïve bayes در این دیتاست بیشتر بوده است.

## قسمت ب)

حال می‌خواهیم مشابه با همین کار برای شبکه‌های عصبی از نوع بازگشتی LSTM پیش‌بینی داده متنی را انجام دهیم. برای اینکار لازم است داده بصورت sequential بشود که از توابع pad_sequence لازم است استفاده کنیم. بصورت کلی از توابع keras و Tensorflow استفاده خواهیم کرد.

ابتدا لازم است تا با استفاده از تابع پیش‌پردازش متن داده‌ها را تمیز کرده و سپس وارد بخش sequence بشویم. برای این کار از کد زیر استفاده می‌کنیم.

```
x = actualrating['review_body'].reset_index()
X = x['review_body']
review_cleans = [preprocess_text(x) for x in X];
sentences = [' '.join(r) for r in review_cleans ]
```

حال لازم است تا هر کلمه را بصورت tokenize دربیاوریم. برای اینکار از تابع Tokenizeer و fit_on_texts در کتابخانه keras استفاده می‌کنیم. همچنین در ادامه نیز این توکن‌ها را بصورت sequenceای از اعداد با تابع texts_to_sequence درمی‌آوریم و به یک بردار از نوع numpy تبدیل می‌کنیم. همچنین ایندکس آن‌ها را نیز ذخیره می‌کنیم و یک دیکشنری کلی تشکیل می‌دهیم.

```
from keras.preprocessing.text import Tokenizer
import numpy as np

#Keras
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)

text_sequences = np.array(tokenizer.texts_to_sequences(sentences))
sequence_dict = tokenizer.word_index
word_dict = dict((num, val) for (val, num) in sequence_dict.items())
```

در نهایت مجموعه‌ای از کل sequnceها را در یک حلقه ایجاد می‌کنیم و به تابع pad_sequence ورودی می‌دهیم. این تابع در واقع برای ما اطمینان حاصل می‌کند که همه sequenceها دارای یک اندازه هستند و اگر نباشد با اعدادی پر می‌کند. مقدار maxlen نیز یک مقدار مهم برای این تابع است که در اینجا ما برابر با ۲۰ قرار داده‌ایم.

```
from keras.utils import pad_sequences

max_cap =20;
reviews_encoded = [];
for i,review in enumerate(review_cleans):
    reviews_encoded.append([sequence_dict[x] for x in review]);

X = pad_sequences(reviews_encoded, maxlen=max_cap, truncating='post')
```

حال ۶۰ درصد داده‌های ابتدایی را برای آموزش و ۲۰ درصد برای validation و ۲۰ درصد آخری را برای تست جدا می‌کنیم. همچنین برای داده‌های خروجی نیز اگر label برابر با ۱ بود مقدار ۰ و درغیر این صورت برابر ۱ می‌شود:

```
Y = np.array([0 if label==1 else 1 for label in y])

np.random.seed(1024);
random_posits = np.arange(len(X))
np.random.shuffle(random_posits);

X = X[random_posits];
Y = Y[random_posits].reshape([-1,1]);

train_cap = int(0.6 * len(X));
dev_cap = int(0.8 * len(X));

X_train, Y_train = X[:train_cap], Y[:train_cap]
X_dev, Y_dev = X[train_cap:dev_cap], Y[train_cap:dev_cap]
X_test1, Y_test1 = X[dev_cap:], Y[dev_cap:]
```

حال لازم است تا شبکه عصبی مورد نظر را پیاده‌سازی کنیم. برای اینکار ابتدا یک لایه Embedding استفاده می‌کنیم. این لایه باعث می‌شود تا ایندکس‌ه ا به یک سری وکتور با اندازه یکسان تبدیل شوند. در حقیقت این موقعیت‌ها جایگاه کلمه در متن می‌باشد. ابعاد ورودی نیز برابر با تعداد کلمات می‌باشد. همچنین ابعاد خروجی نیز برابر با max_cap که برابر با ۲۰ درنظر گرفته بودیم می‌باشد. سپس یک لایه LSTM با ۱۵۰ نورون و یک لایه دیگر با همین مقدار نورون قرار می‌دهیم. بعد از این لایه یک لایه شبکه dense با ۱۰۰ نورون و در نهایت نیز یک لایه با یک نورون برای Classification قرار می‌دهیم. بهینه‌ساز را برابر با Adam قرار می‌دهیم. تعداد ایپاک برابر با ۵۰ و بچ سایز را برابر با ۶۴ قرار می‌دهیم.

```
model1 = Sequential();
model1.add(Embedding(len(word_dict)+1, max_cap, input_length=max_cap));
#adding a LSTM layer of dim 1--
model1.add(LSTM(150, return_sequences=True));
model1.add(LSTM(150, return_sequences=False));
#adding a dense layer with activation function of relu
model1.add(Dense(100, activation='relu'));#best 50,relu
#adding the final output activation with activation function of softmax
model1.add(Dense(1, activation='sigmoid'));
print(model1.summary());
optimizer = Adam(lr=0.0001, decay=0.0001);

model1.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
# fit model and run it for 5 epochs
model1.fit(X_train, Y_train, batch_size=64, epochs=50, validation_data=(X_dev, Y_dev))
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 20, 20)            499600

 lstm (LSTM)                 (None, 20, 150)           102600

 lstm_1 (LSTM)               (None, 150)               180600

 dense (Dense)               (None, 100)               15100

 dense_1 (Dense)             (None, 1)                 101
```

خروجی شبکه بصورت زیر خواهد بود:

```
[45] model1_preds = model1.predict(X_test1)
     model1_preds[model1_preds > 0.5] = 1
     model1_preds[model1_preds <= 0.5] = 0
     print(confusion_matrix(Y_test1, model1_preds))
     print('\n')
     print(classification_report(Y_test1, model1_preds))
```

```
428/428 [==============================] - 1s 3ms/step
[[ 1134   627]
 [  396 11512]]


              precision    recall  f1-score   support

           0       0.74      0.64      0.69      1761
           1       0.95      0.97      0.96     11908

    accuracy                           0.93     13669
   macro avg       0.84      0.81      0.82     13669
weighted avg       0.92      0.93      0.92     13669
```

**گزارش پروژه**

## قسمت ج)

همانطور که مشاهده می‌شود دقت شبکه عصبی از دقت مدل‌های ماشین لرنینگ کمتر شده‌است. اما پارامترهای بسیار زیادی در شبکه‌های عصبی وجود دارند که می‌توان با بهینه‌سازی آن‌ها دقت شبکه را بالاتر برد و پتانسیل افزایش دقت بسیار بیشتری نسبت به مدل‌های ماشین لزنینگ دارند. این پارامترها شامل max_cap، تعداد نورون‌های شبکه، تعداد لایه‌های شبکه، معماری شبکه، تعداد ایپاک شبکه، تعداد batch size های شبکه، نرخ یادگیری و نوع بهینه‌ساز می‌باشد. با بهینه‌سازی یا تغییر بصورت تجربی این پارامترها به راحتی می‌توان به دقت بالاتر از مدل‌های ماشین لرنینگ دست پیدا کرد.

## قسمت امتیازی BERT Embedding

در این قسمت قصد داریم تا با استفاده از Transfer Learning و مدل BERT که از قبل آموزش دیده است، طبقه‌بندی این دیتاست را انجام دهیم. برای اینکار از پکیج HuggingFace که در دسته Transformerها قرار دارد استفاده می‌کنیم.

ابتدا لازم است تا کتابخانه‌های لازم را importکنیم:

سپس لازم است تا دیتاست را متناسب با این مدل آماده سازی کنیم. مکانیزم توکن‌سازی این مدل نیز متناسب با خودش است و لازم است تا فرایندهای پیش‌پردازش آن را انجام دهیم. بنابراین برای vectorization متن از توکنایزر مخصوص به مدل استفاده می‌کنیم. این مدل از توکنایزر WordPeice استفاده می‌کند.

برای این مدل از BertTokenizer.encode_plus() برای تبدیل sequence ها به فرمت‌های ورودی برای طبقه‌بندی داده‌ها استفاده می‌کنیم. این تابع سه خروجی input_dis که مشابه تابع text_to_sequences در کراس است، type_token_ids که این ids به شماره جملات که توکن‌ها به آن تعلق دارد می‌باشد و attention_mask که مشابه با Mask در کراس است و نشان می‌دهد که توکن‌ها، توکن‌های واقعی هستند.

حال بر اساس این توضیحات می‌خواهیم تا متن‌ها را به ورودی این مدل تبدیل کنیم. برای اینکار از کد زیر استفاده می‌کنیم.

```
num_classes = 2
bert_tokenizer = BertTokenizer.from_pretrained("bert-base-uncased", do_lower_case=True)

def convert_fun_to_feature(review):
  return bert_tokenizer.encode_plus(review,
                add_special_tokens = True,
                max_length = 32,
                padding='max_length',
                truncation=True,
                return_attention_mask = True,
              )
def map_fun_to_dict(input_ids, attention_masks, token_type_ids, label):
  return {
      "input_ids": input_ids,
      "token_type_ids": token_type_ids,
      "attention_mask": attention_masks,
  }, label

def encode_fun(ds):
  input_ids_list = []
  token_type_ids_list = []
  attention_mask_list = []
  label_list = []
  for review, label in ds:
    bert_input = convert_fun_to_feature(review)
    input_ids_list.append(bert_input['input_ids'])
    token_type_ids_list.append(bert_input['token_type_ids'])
    attention_mask_list.append(bert_input['attention_mask'])
    label_list.append([label])

  return tf.data.Dataset.from_tensor_slices((input_ids_list, attention_mask_list, token_type_ids_list, label_list)).map(map_fun_to_dict)
```

بر اساس این کد تابع اصلی که ورودی را دریافت می‌کند encode_fun است. ابتدا دیتای ورودی را دریافت کرده و سپس برای هر خروجی تابع bert_tokenizer.encode_plut یک ماتریس تشکیل می‌دهد. سپس در یک حلقه‌ای که شامل داده ورودی و برچسب داده است تابع bert_tokenizer.encode_plus را صدا می‌زند و خروجی‌های آن را در ماتریس‌ها می‌ریزد. بعد از این حلقه در نهایت یک دیکشنری که شامل خروجی‌های تابع می‌باشد را بر می‌گرداند.. این تابع به شکل زیر صدا زده می‌شود:

```
batch_size = 64

ds_train = zip(X_train, y_train)
ds_test = zip(X_test, y_test)
ds_train_encoded = encode_fun(ds_train).shuffle(len(X_train)).batch(batch_size)
ds_test_encoded = encode_fun(ds_test).batch(batch_size)
```

پس از آنکه وردی‌ها را برای این شبکه آماده کردیم لازم است تا مدل را بسازیم. ابتدا یک محل برای ذخیره مدل‌سازیم و سپس خود مدل را با تابع TFBertForSequenceClassification می‌سازیم:

```
model_save_path = './bert_model.h5'

path = "./models/"

bert_model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=num_classes)

bert_model.summary()
```

```
All PyTorch model weights were used when initializing TFBertForSequenceClassification.

Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification were not initialized from the P
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inferen
Model: "tf_bert_for_sequence_classification"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bert (TFBertMainLayer)      multiple                  109482240

 dropout_37 (Dropout)        multiple                  0

 classifier (Dense)          multiple                  1538

=================================================================
Total params: 109,483,778
Trainable params: 109,483,778
Non-trainable params: 0
_________________________________________________________________
```

حال لازم است تا مقدار نرخ یادگیری، تعداد ایپاک، بهینه‌ساز و تابع هزینه را به مدل ورودی دهیم:

```
learning_rate = 2e-5

number_of_epochs = 2

optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate, epsilon=1e-08)

loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')

bert_model.compile(loss=loss,
                   optimizer=optimizer,
                   metrics=metric)
```

حال با تابع fit مدل را آموزش می‌دهیم:

```
history = bert_model.fit(ds_train_encoded,
                         batch_size=batch_size,
                         epochs=number_of_epochs,
                         validation_data=ds_test_encoded)
```

```
Epoch 1/2
746/746 [==============================] - 392s 459ms/step - loss: 0.4441 - accuracy: 0.8653 - val_loss: 0.3947 - val_accuracy: 0.8659
Epoch 2/2
746/746 [==============================] - 338s 452ms/step - loss: 0.4268 - accuracy: 0.8649 - val_loss: 0.3973 - val_accuracy: 0.8659
```

## گزارش پروژه

خروجی این مدل پس از ۲ ایپاک بصورت زیر است. همانطور که از تصویر زیر مشاهده می‌شود مقدار دقت این شبکه برای داده مذکور از شبکه‌ای که با استفاده از LSTM ها و مدل‌های ماشین لرنینگ ساخته شد کمتر است. اما می‌توان با متناسب سازی داده‌ها و تکنیک‌های تمیز کردن داده و... دقت این مدل را نیز افزایش داد.

```python
def plot1(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)
    plt.plot(epochs, acc, 'bo', label='Training acc')
    plt.plot(epochs, val_acc, 'b', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.legend()
    plt.figure()
    plt.plot(epochs, loss, 'bo', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()
    plt.show()

def plot2(history):
    pd.DataFrame(history.history).plot(figsize=(4, 2))
    plt.grid(True)
    plt.show()
plot2(history)
```