Lecture 23

Text Seatmate Analysis

Python code to use ML to do intimate analysis.

```
#!/Users/philip/opt/anaconda3/bin/python

# TensorFlow 2.7
# From: https://www.tensorflow.org/tutorials/keras/text_classification

import matplotlib.pyplot as plt
import os
import re
import shutil
import string
import tensorflow as tf

from tensorflow.keras import layers
from tensorflow.keras import losses

print("TensorFlow Version: {}".format(tf.__version__))

url = "https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"

dataset = tf.keras.utils.get_file("aclImdb_v1", url,
                                    untar=True, cache_dir='.',
                                    cache_subdir='')

dataset_dir = os.path.join(os.path.dirname(dataset), 'aclImdb')

os.listdir(dataset_dir)

train_dir = os.path.join(dataset_dir, 'train')
os.listdir(train_dir)

sample_file = os.path.join(train_dir, 'pos/1181_9.txt')
print ( "PJS: input sample: {}".format(sample_file) )

with open(sample_file) as f:
    print(f.read())

remove_dir = os.path.join(train_dir, 'unsup')
shutil.rmtree(remove_dir)

batch_size = 32
seed = 42

```

```
raw_train_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/train',
    batch_size=batch_size,
    validation_split=0.2,
    subset='training',
    seed=seed)

for text_batch, label_batch in raw_train_ds.take(1):
    for i in range(3):
        print("Review", text_batch.numpy()[i])
        print("Label", label_batch.numpy()[i])

print("Label 0 corresponds to", raw_train_ds.class_names[0])
print("Label 1 corresponds to", raw_train_ds.class_names[1])

raw_val_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/train',
    batch_size=batch_size,
    validation_split=0.2,
    subset='validation',
    seed=seed)

raw_test_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/test',
    batch_size=batch_size)


def custom_standardization(input_data):
    lowercase = tf.strings.lower(input_data)
    stripped_html = tf.strings.regex_replace(lowercase, '<br />', ' ')
    return tf.strings.regex_replace(stripped_html, '[%s]' % re.escape(string.punct


max_features = 10000
sequence_length = 250

vectorize_layer = layers.TextVectorization(
    standardize=custom_standardization,
    max_tokens=max_features,
    output_mode='int',
    output_sequence_length=sequence_length)

# Make a text-only dataset (without labels), then call adapt
train_text = raw_train_ds.map(lambda x, y: x)
vectorize_layer.adapt(train_text)

def vectorize_text(text, label):
    text = tf.expand_dims(text, -1)
    return vectorize_layer(text), label

# retrieve a batch (of 32 reviews and labels) from the dataset
```

```
text_batch, label_batch = next(iter(raw_train_ds))
first_review, first_label = text_batch[0], label_batch[0]
print("Review", first_review)
print("Label", raw_train_ds.class_names[first_label])
print("Vectorized review", vectorize_text(first_review, first_label))

print("1287 ---> ",vectorize_layer.get_vocabulary()[1287])
print(" 313 ---> ",vectorize_layer.get_vocabulary()[313])
print('Vocabulary size: {}'.format(len(vectorize_layer.get_vocabulary())))


train_ds = raw_train_ds.map(vectorize_text)
val_ds = raw_val_ds.map(vectorize_text)
test_ds = raw_test_ds.map(vectorize_text)

AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)

embedding_dim = 16


model = tf.keras.Sequential([
    layers.Embedding(max_features + 1, embedding_dim),
    layers.Dropout(0.2),
    layers.GlobalAveragePooling1D(),
    layers.Dropout(0.2),
    layers.Dense(1)])

model.summary()


model.compile(loss=losses.BinaryCrossentropy(from_logits=True),
              optimizer='adam',
              metrics=tf.metrics.BinaryAccuracy(threshold=0.0))

epochs = 10
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs)

loss, accuracy = model.evaluate(test_ds)

print("Loss: ", loss)
print("Accuracy: ", accuracy)

history_dict = history.history
history_dict.keys()
```

```

acc = history_dict['binary_accuracy']
val_acc = history_dict['val_binary_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']

epochs = range(1, len(acc) + 1)

# "bo" is for "blue dot"
plt.plot(epochs, loss, 'bo', label='Training loss')
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()


plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

plt.show()

# --------------------------------------------------------------------------------

export_model = tf.keras.Sequential([
  vectorize_layer,
  model,
  layers.Activation('sigmoid')
])

export_model.compile(
    loss=losses.BinaryCrossentropy(from_logits=False), optimizer="adam", metrics=[
)

# Test it with `raw_test_ds`, which yields raw strings
loss, accuracy = export_model.evaluate(raw_test_ds)
print(accuracy)

examples = [
  "The movie was great!",
  "The movie was okay.",
  "The movie was terrible..."
]

```

```
195: print ( export_model.predict(examples) )
```

What is “Mechanical Turk”

https://en.wikipedia.org/wiki/Mechanical_Turk

How Google Search Works

Google started out searching pages based on links to pages. This was the page link algorithm.

People figured out how to cheat.

Google moved to a more robust system that analyzed the text content in the pages.

SEO optimization was born. In other words people figured out how to cheat.

Now Google uses a “page ranking” system. 70% of requests/searches that Google sees are new - it has never seen that request before. But it can use ML to compare that search to similar searches in the past. It can also compare these previous searches to the similarity to pages. But the question is how to rank the pages so that the most probable answer is first.

That is easy - what did people click on? When did they stop clicking? The got the answer that they wanted when they stooped clicking. (Or they got too frustrated and did a new search)

Now how to keep your index fresh. Simple - when you have a new page that is similar to an old page throw it into the mix and see if that is the new “best” answer. If people stop clicking on that new page then it’s ranking will move up.

Each click is a “vote” for the correct answer - based on the human popularity of the answer. The people are ranking the pages.

So… As a smart educated person - having had some ML class - how can you put this to your advantage. Remember all you have to do to win is to be less stupid than all the competitors - in business - in life. You don’t need to be smarter. Just less stupid.

1. Is somebody paying - or making something of value - in cheating.
2. Can the system be easily gamed. Can they cheat.
3. is there 0 benefit to cheating.
4. is the source of the answer impartial.
5. are the “biases” of the source in a direction that will not influence the accuracy of the answer.

So how do you get Google to index your page and find it today.

1. Provide good useful content to the humans.
2. Put Google advertising on the page.
3. Add in youtube video links on the page.

Always remember that whatever you Google the results are just a popularity contest

Predicting Housing Prices.

Some experience in this.

Some experience in Finance also.

Zillow Market Price. I compared that to my naive model, sqft, bedrooms, bathrooms, kitchen, remodeled, location, average price of neighbors etc. Just 20 factors and then compare that to actual purchase price and number of days on market.

Number of days on market is a good indicator of over priced. My naive model was doing better than the Zillow "price" model.

Now that Zillow has crashed, 25% of staff cut, entire "Data Science" team dropped, stock down 30%, we can look back on what they were doing and make some observations. This is the entire "Zillow Offers" section of the business. Gone - $500 million oops.

1. Field Knowledge is required.
2. Knowing what is "hard" is a good idea. Stocks, Markets, Real-Estate, Futures are all "stochastic" markets.
3. Don't just believe you know more than existing "experts".

This is what Zillow did. They created a "new" business model with a high level of confidence that they could predict the price that a home was worth. They could also predict the price that it would be worth in 6 months. Then if the margin was sufficiently high they would offer, purchase, hold and sell in 6 months. This is in a rising market! To get the expected returns the model had to be super accurate! To justify the cost of 2500 employees they had to take this to scale right away. Lots and Lots of homes!

Then you saw ads for staph for the Zillow Offers data team that required knowledge of "Prophet" a Python package that performs seasonal time series data fitting. This is multivariate linear correlation with time period seasonal variations removed. This is *not* some complex machine learning system. Turns out that the "Zillow home price" calculation was just a linear model fit.

If you want to use a practical machine learning model you build the model, then you test the perdition results on a statistical sample of houses - maybe as low as 10 or 15 and then you refine your model and asses your level of risk. Look at WayMo(Google) and self driving cars. They build the model then they drove 23 vehicles around with a safety driver for 80,000 miles, then they upped the any to 50 vehicles and 200,000 miles, then 500,000 miles, then 1,000,000 miles. Each time testing the results and refining the model. Then they were behind schedule by a year - and they kept at it for 3 more years! Before they released a real result they had 29,000,000 miles and 140,000 vehicles! 140,000 in a single city! The goal is fully-autonomous driving for the world. 10 years in and they are working on the 2nd city! AND - AND this is important - this is an easier problem than a “stochastic” problem!

SO.. What is stochastic?

Stochastic:

Randomly determined; having a random probability distribution or pattern that may be analyzed statistically but may not be predicted precisely.

Now remember that Zillow Offers had 0 reactors, 0 professional property managers, 0 fiance people, 0 people from REITs - they had a bunch of data science people that had never actually gone out and “owned” and “managed” property.

When I did “pickle picker” the first thing that I did was get on the line and spend 3 days picking pickles. Talked to the old timers about what the company was going to have them do after picking pickles. Then I talked to the old folks that had been doing it for years. What weird things happen? What sticks out in your memory? Tell me stories. The people on the “floor” and the ones with the “knowledge” of what really goes on - they are the “experts”. It makes no difference if they only have an 8th grade education - they know a process, a system and know all the ins and outs of the system. If you are going to automate that system you need that knowledge.

Stocks are “stochastic” - If I do a good job picking - 5 stocks. 1 will make me mad, 3 will do about what I predict, 1 will make me happy. Same for houses. I can predict on average that the rate of return of good picks will be about 10%. This is not some huge earth shattering return. This is a decent return for an invested amount of money. Houses I can increase the value of the property by getting a house, fixing it up, and then selling it. That requires skill and time. It requires construction knowledge. It requires area knowledge.

It requires “field” knowledge.

So... How do you apply this in your world.

You get some job that is using ML - be happy. Pay is good. Start polkaing into the questions.

- Who are the field knowledge experts?

- How is this tested?
- Is management in an echo chamber?
- Is this a temporary job or a real opportunity?
- Are the “plans” realistic?

Small Alterations to Signs result in Misc Classification

https://spectrum.ieee.org/slight-street-sign-modifications-can-fool-machine-learning-algorithms