Seminar work 4IZ172 – Text analytics 1, by Maksim Obukhov

The whole work is performed using Python and Jupyter Notebook

Dataset description

After long consideration of what dataset to choose, I have found The Financial Phrasebank dataset at Hugging Face webpage, which contains sentences from financial news. The dataset consists of 3453 sentences from English language financial news categorized by sentiment. The dataset is divided by agreement rate of 5-8 annotators. Author of the dataset offers to download several versions: 50%, 66%, 75%, 100% of the annotators agreement. Through the work I use 75% version. The dataset is available in the Hugging Face datasets library and can be used for sentiment classification task. The dataset is suitable for tasks such as text classification, specifically sentiment classification, and is available in English. The source data was written by various financial journalists and the annotations were generated by experts. The dataset is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 license.

Attributes

The dataset is represented as a `.txt` file containing plain text, where each line is one sentences followed by @-sign and corresponding label. Here's an example of first 5 rows:

- According to Gran , the company has no plans to move all production to Russia , although that is where the company is growing .@**neutral**
- With the new production plant the company would increase its capacity to meet the expected increase in demand and would improve the use of raw materials and therefore increase the production profitability .@**positive**
- For the last quarter of 2010 , Componenta 's net sales doubled to EUR131m from EUR76m for the same period a year earlier , while it moved to a zero pre-tax profit

from a pre-tax loss of EUR7m .@**positive**

- In the third quarter of 2010 , net sales increased by 5.2 % to EUR 205.5 mn , and operating profit by 34.9 % to EUR 23.5 mn .@**positive**
- Operating profit rose to EUR 13.1 mn from EUR 8.7 mn in the corresponding period in 2007 representing 7.7 % of net sales .@**positive**

Label corresponding to the class as a string:

- 'positive'
- 'negative'
- 'neutral'

Sentiment classification

Task description

Sentiment classification is the automated process of identifying opinions in text and labeling them as positive, negative, or neutral based on the emotions expressed within them[1]. The task involves using natural language processing (NLP) and machine learning techniques to interpret subjective data and determine the overall sentiment conveyed by a particular text, phrase, or word[2].

There are several methods to conduct sentiment analysis, including rule-based and automated sentiment analysis, lexicon-based approaches, classical machine learning algorithms, and deep learning algorithms[3]. In this work we use **BERT language model** (more specified later).

Citations:
[1] https://monkeylearn.com/blog/sentiment-classification/
[2] https://getthematic.com/sentiment-analysis/
[3] https://research.aimultiple.com/sentiment-analysis-methods/

Solution and particular code cells

More detailed solution and the whole code (jupyter notebook) to be found in `.zip` *file provided together*

The first step of the work is importing **libraries**. Some important libraries we use:

- Pandas
- NumPy
- Matplotlib
- Scikit-learn
- Transformers
- TensorFlow
- Datasets
- Wordcloud

Dataset preparing

Dataset building

The dataset is represented as a plain txt file. After loading the file and converting into a pandas DataFrame type, we divide the first column by “@”, and obtain two columns “text”, containing all the observations of financial news without label, and the second column “label”, containing corresponding labels.

Then we perform label mapping, converting corresponding label to a number:

```
# Mapping dictionary
mapping = {'positive': 1, 'negative': 0, 'neutral': 2}

# Apply mapping to the 'label' column
df['label'] = df['label'].map(mapping)
df.head()
```

Converting into Dataset type

After performing data splitting into training, test and validation as 70%, 20% and 10% respectively, and converting pandas dataframes into DatasetDict type we obtain:

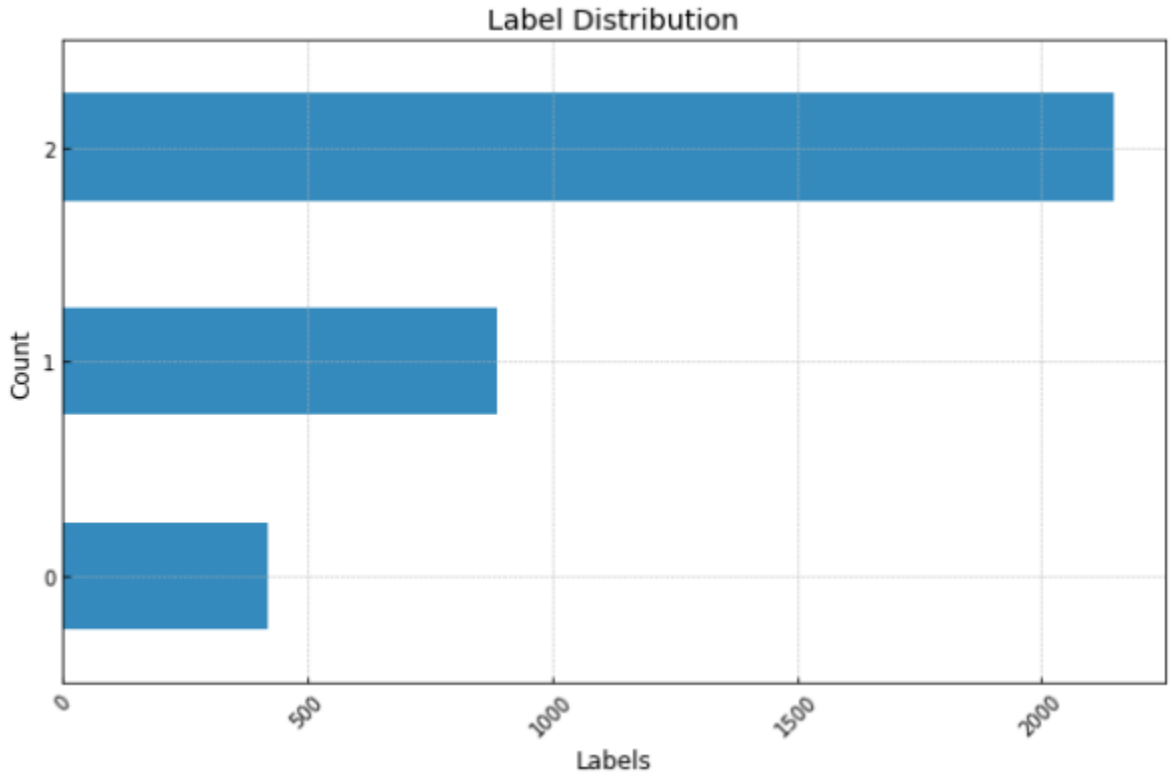
```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
```

```
        num_rows: 2417
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 694
    })
    validation: Dataset({
        features: ['text', 'label'],
        num_rows: 342
    })
})
```

Data understanding

Label distribution

First, we have to know what's the label distribution in our dataset. Using matplotlib obtain a chart:



Where labels do not equally distributed. Most of the instances are neutral (2146 instances). Positive labels are in the middle (887 instances). Negative label is the least

frequent (420 instances).

Word clouds

Positive labels:

We can see the most popular words representing positive sentiment such as:

- profit
- increase
- growth
- won
- sales

Negative labels:

Similarly, the most frequent negative words:

- loss
- fell
- decreased
- declined
- dropped

Neutral labels:

Similarly, the most frequent neutral words:

- include
- will
- said
- share
- new

Data preprocessing

Before we start training a model, we have to perform preprocessing steps: batch and tokenize sentences to be analyzed, creating a collator, and converting dataset into TensorFlow dataset types.

Tokenization

We perform tokenization using AutoTokenizer with model `"distilbert-base-uncased"`

```
tokenizer = AutoTokenizer.from_pretrained(pretrained_model_name_or_path="distilbert-base-u
ncased", max_length=512)

def tokenize(batch):
    return tokenizer(batch["text"], padding=True, truncation=True)

tokenized_dataset = dataset.map(tokenize, batched=True, batch_size=None)
tokenized_dataset.with_format('tensorflow')
```

We obtain two new columns “input_ids” and “attention_mask”. The output:

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label', 'input_ids', 'attention_mask'],
        num_rows: 2417
    })
    test: Dataset({
        features: ['text', 'label', 'input_ids', 'attention_mask'],
        num_rows: 694
    })
    validation: Dataset({
        features: ['text', 'label', 'input_ids', 'attention_mask'],
        num_rows: 342
    })
})
```

Example:

```
>>print(tokenized_dataset["train"][:1])
{'text': ['The expanded company will continue to be called NewPage .'], 'label': [2], 'inp
ut_ids': [[101, 1996, 4423, 2194, 2097, 3613, 2000, 2022, 2170, 2047, 13704, 1012, 102, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]], 'attention_mas
k': [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]}
```

Data collator and TensorFlow dataset type

We specify batch size as 64, create `DataCollatorWithPadding` and convert tokenized datasets into TF type

```
batch_size = 64
data_collator = DataCollatorWithPadding(tokenizer=tokenizer, return_tensors="tf")

tf_train_dataset = tokenized_dataset["train"].to_tf_dataset(
    columns=["input_ids", "attention_mask"],
    label_cols=["label"],
    shuffle=True,
    batch_size=batch_size,
    collate_fn=data_collator
)

tf_valid_dataset = tokenized_dataset["validation"].to_tf_dataset(
    columns=["input_ids", "attention_mask"],
    label_cols=["label"],
    shuffle=False,
    batch_size=batch_size,
    collate_fn=data_collator
)

tf_test_dataset = tokenized_dataset["test"].to_tf_dataset(
    columns=["input_ids", "attention_mask"],
    label_cols=["label"],
    shuffle=False,
    batch_size=batch_size,
    collate_fn=data_collator
)
```

Model initialization

As a classification model choosed `TFAutoModelForSequenceClassification` with `from_pretrained` method, where we specify a number of labels, in our case 3.

```
model = TFAutoModelForSequenceClassification.from_pretrained(model_ckpt, num_labels=3)
model.compile(
    optimizer=optimizers.Adam(learning_rate=5e-5),
    loss=losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=metrics.SparseCategoricalAccuracy())
model.summary()

Output:
Model: "tf_distil_bert_for_sequence_classification"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 distilbert (TFDistilBertMai  multiple                 66362880
```

```
 nLayer)

 pre_classifier (Dense)      multiple                  590592

 classifier (Dense)          multiple                  2307

 dropout_19 (Dropout)        multiple                  0

=================================================================
Total params: 66,955,779
Trainable params: 66,955,779
Non-trainable params: 0
_________________________________________________________________
```

Training the model

We train the model fitting train dataset and validation, where attribute `epochs` is specified as 5. Training the model takes on my computer about 10 minutes per epoch, so it is about an hour to train our model. Training process is performed on **CPU**.

```
model.fit(tf_train_dataset,
          validation_data=tf_valid_dataset,
          epochs=5)
```

In case we don’t want to train the model each time we run the whole code, we can save weights and load them later.

```
model.save_weights('my_model.h5')

model = create_model()
model.load_weights('my_model.h5')
```

Testing the model

Testing a model is crucial. We want to know how good the model performs predictions. As we have already divided our dataset into 3 parts, where the test part contains 20% (694 obs.), we perform testing on the test dataset.

```
input = "BTC price decreases"
outputs = model.predict(tokenizer(input)["input_ids"])
outputs['logits'][0].tolist()
label_int = np.argmax(tf.keras.layers.Softmax()(outputs['logits'][0].tolist()))
```

```
defget_mapping_value(value):
forkey, mapped_valueinmapping.items():
ifmapped_value == value:
returnkey
return None

print(get_mapping_value(label_int.item()))
```

We obtain result:

```
1/1 [==============================] - 0s 50ms/step
negative
```

The prediction is correct.

Model evaluation

As one more metric of model evaluation is measuring accuracy of predictions.

```
results = model.evaluate(tf_test_dataset)
print("Test set accuracy: {:.2f}%".format(results[1] * 100))

Output:
11/11 [==============================] - 18s 1s/step - loss: 0.2460 - sparse_categorical_a
ccuracy: 0.9222
Test set accuracy: 92.22%
```

We can conclude that the model correctly predicted the label 92.22% of instances. It performs very well. The task is completed.

Topic modeling

Task description

Topic modeling is a text-mining technique that involves identifying topics that best describe a set of documents. It is a statistical process through which you can identify, extract, and analyze topics from a given collection of documents[1]. The technique is used to uncover hidden patterns and thematic structures within a collection of documents and to derive hidden patterns exhibited by a text corpus[2].

Topic modeling can be used to understand the main themes present in a collection of documents and to cluster similar documents together. It is an unsupervised approach used for finding and observing the bunch of words (called “topics”) in large clusters of texts. The process of learning, recognizing, and extracting these topics across a collection of documents is called topic modeling[3].

There are several techniques for topic modeling, including Latent Dirichlet Allocation (LDA), Non-Negative Matrix Factorization (NMF), Latent Semantic Analysis (LSA), Parallel Latent Dirichlet Allocation (PLDA), and Pachinko Allocation Model (PAM). LDA is one of the most popular topic modeling techniques and is used to identify topics that emerge during the topic modeling process[4].

Topic modeling can be used in various industries, including bioinformatics, social media monitoring, email spam filtering, chatbots, autocorrection, speech recognition, language translation, hiring and recruitment, and market research. It can also be used for text classification, categorization, and summarization of documents.[5]

Citations:
[1] https://medium.com/analytics-vidhya/topic-modelling-techniques-37826fbab549
[2] https://www.analyticsvidhya.com/blog/2016/08/beginners-guide-to-topic-modeling-in-python/
[3] https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05
[4] https://www.analyticsvidhya.com/blog/2021/05/topic-modelling-in-natural-language-processing/
[5] https://medium.com/@fatmafatma/industrial-applications-of-topic-model-100e48a15ce4

Solution and particular code cells

The first step of the work is importing **libraries**. Some important libraries we use:

- nltk
- Scikit-learn

Corpus preparation

Before performing the task we have to prepare a corpus by:

- Removing stop words
- Removing numbers
- Stemming

```
# Remove stop words
stop_words = set(stopwords.words('english'))
corpus = df['text'].apply(lambda x: ' '.join([word for word in word_tokenize(x) if word.lo
wer() not in stop_words]))

# Create a stemmer object
stemmer = PorterStemmer()

# Perform stemming
stemmed_corpus = []
for text in corpus:
    tokens = word_tokenize(text)  # Tokenize the text into individual words
    stemmed_tokens = [stemmer.stem(token) for token in tokens]  # Perform stemming on each
token
    stemmed_text = ' '.join(stemmed_tokens)  # Join the stemmed tokens back into a string
    stemmed_corpus.append(stemmed_text)

# Remove numbers
stemmed_corpus = [re.sub(r'\d+', '', text) for text in stemmed_corpus]

print(stemmed_corpus[:2])

output:
['accord gran , compani plan move product russia , although compani grow .', 'new product
 plant compani would increas capac meet expect increas demand would improv use raw materi
 therefor increas product profit .']
```

Document term matrix

Then creating a document term matrix of the stemmed corpus:

```
count_vect = CountVectorizer(max_df=0.8, min_df=2)
doc_term_matrix = count_vect.fit_transform(stemmed_corpus)
```

We specify to only include those words that appear in less than 80% of the document and appear in at least 2 documents.

LDA model

As mentioned above, the most common topic modeling function is `LatentDirichletAllocation`.

```
LDA = LatentDirichletAllocation(n_components=5, random_state=42)
LDA.fit(doc_term_matrix)
```

The parameter `n_components` specifies the number of categories, or topics, that we want our text to be divided into, in our case it is 3. The parameter `random_state` is set to 42 so that we get the same results.

Results

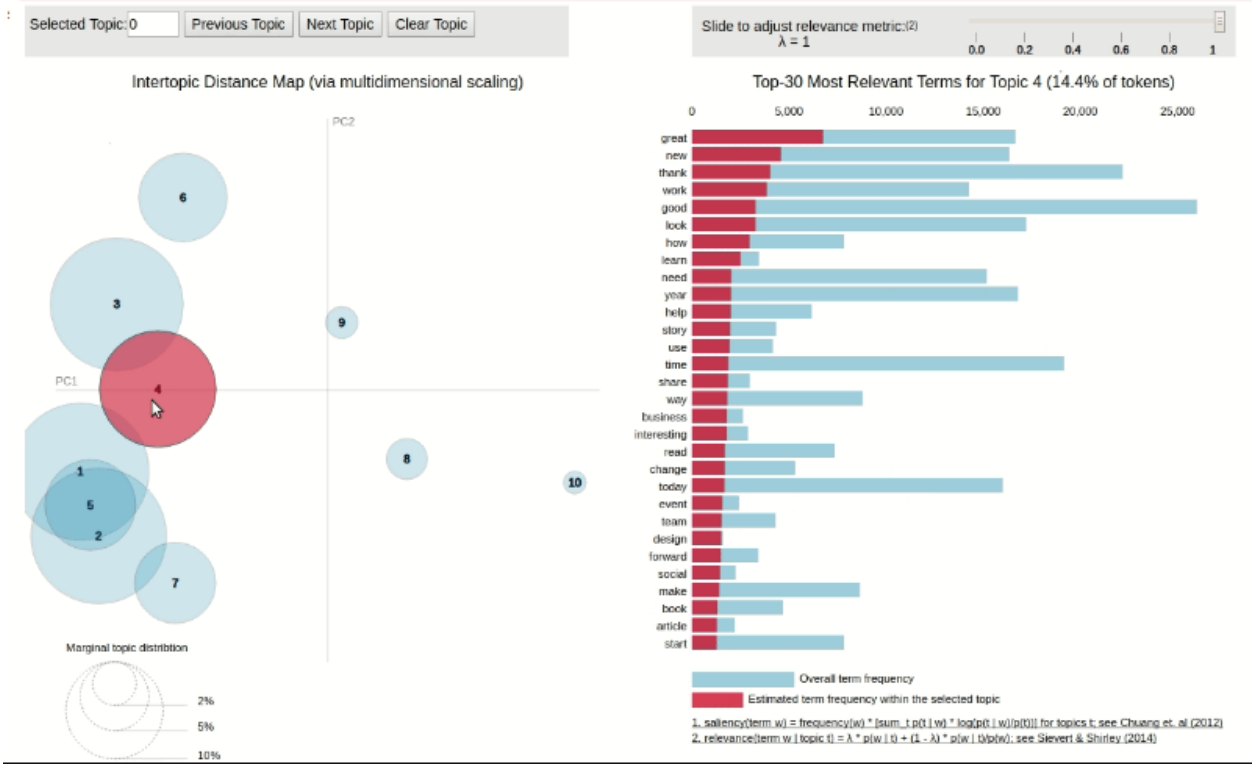
We obtain 3 topics and 5 the most popular words per topic:

```
Top 5 words for topic #0:
['oyj', 'contract', 'said', 'finnish', 'servic']


Top 5 words for topic #1:
['sale', 'net', 'profit', 'mn', 'eur']


Top 5 words for topic #2:
['share', 'oper', 'product', 'also', 'compani']


Top 5 words for topic #3:
['finnish', 'euro', 'compani', 'mln', 'share']


Top 5 words for topic #4:
['nokia', 'finland', 'oper', 'start', 'compani']
```

Visualization

Visualization is performed using `pyLDAvis` library.

```
vis_data = pyLDAvis.sklearn.prepare(lda_model, corpus, dictionary)
pyLDAvis.display(vis_data)
```



Collocation

Task description

Collocation extraction is the task of using a computer to extract collocations automatically from a corpus. Collocations are defined as a sequence of words or terms that co-occur more often than would be expected by chance[1]. For example, "riding boots" and "motor cyclist" are examples of collocated pairs of words. Collocations can be used to identify patterns and thematic structures within a collection of documents[2].

Citations:
[1] https://en.wikipedia.org/wiki/Collocation_extraction
[2] https://www.britishcouncil.org/voices-magazine/fun-ways-teach-english-collocations

Solution and particular code cells

The first step of the work is importing **libraries**. Some important libraries we use:

- nltk
- Scikit-learn

- collections

Dictionary creation

Create dictionary of our dataset. Using stemmed coprus before and additionally removing punctuation.

```
words = []
for sublist in stemmed_corpus:
    matches = re.findall(r'\w+', sublist)
    words.extend(matches)

print(words[:20])

output:
['accord', 'gran', 'compani', 'plan', 'move', 'product', 'russia', 'although', 'compani',
 'grow', 'new', 'product', 'plant', 'compani', 'would', 'increas', 'capac', 'meet', 'expec
t', 'increas']
```

Bigram creation

Let’s divide words into bigrams and see what pairs are the most frequent:

```
# prints the 10 most common bigrams
colText = nltk.Text(words)
colText.collocations(10)

output:
net sale; oper profit; correspond period; eur million; euro mln; mln
euro; oyj hel; per share; third quarter; omx helsinki
```

```
colBigrams = list(nltk.ngrams(colText, 2))
print("Number of words:", len(words))
print("Number of bigrams:", len(colBigrams))

output:
Number of words: 42190
Number of bigrams: 42189
```

Here we check to make sure the bigram function has gone through and counted the entire text. Having one less n-gram is correct because of the way in which the n-grams

are generated word-by-word in the test above

TF-IDF matrix

Let’s find 25 most frequent terms by TF-IDF score:

```
# Specify TF-IDF vectorizer
vectorizer = TfidfVectorizer()

# Compute the TF-IDF matrix for the corpus
tfidf_matrix = vectorizer.fit_transform(stemmed_corpus)

# Get the feature names (terms)
terms = vectorizer.get_feature_names_out()

# Identify collocations based on TF-IDF scores
collocations = defaultdict(float)
for i, doc in enumerate(corpus):
    feature_index = tfidf_matrix[i, :].nonzero()[1]
    tfidf_scores = zip(feature_index, [tfidf_matrix[i, x] for x in feature_index])

    for term_idx, score in tfidf_scores:
        term = terms[term_idx]
        collocations[term] = max(collocations[term], score)

# Sort collocations by TF-IDF score in descending order
sorted_collocations = sorted(collocations.items(), key=lambda x: x[1], reverse=True)

# Print the top collocations
top_collocations = sorted_collocations[:25]
for collocation in top_collocations:
    print(collocation)

output, where the first word is a term, and its TF-IDF score:
('forecast', 1.0)
('could', 1.0)
('loan', 1.0)
('welcom', 1.0)
('think', 1.0)
('ls', 0.9461490187934268)
('thousand', 0.9147372572392014)
('sekm', 0.885271235482454)
('capman', 0.885127636876205)
('xa', 0.8571208251401715)
('aspo', 0.8548037030857015)
('catalyst', 0.8535141508484155)
('nd', 0.8377859485034687)
('kemira', 0.8310511568377097)
('billion', 0.8284977537700866)
('mln', 0.8224237824962862)
('onlin', 0.822311325120984)
```

```
('digia', 0.8183396283172262)
('dopplr', 0.8175189047653515)
('nokia', 0.8114146298096994)
('appoint', 0.8080702049604045)
('cap', 0.8012149309251502)
('nwc', 0.8012052708494896)
('eurm', 0.8011543501013417)
('sek', 0.8005964749561502)
```