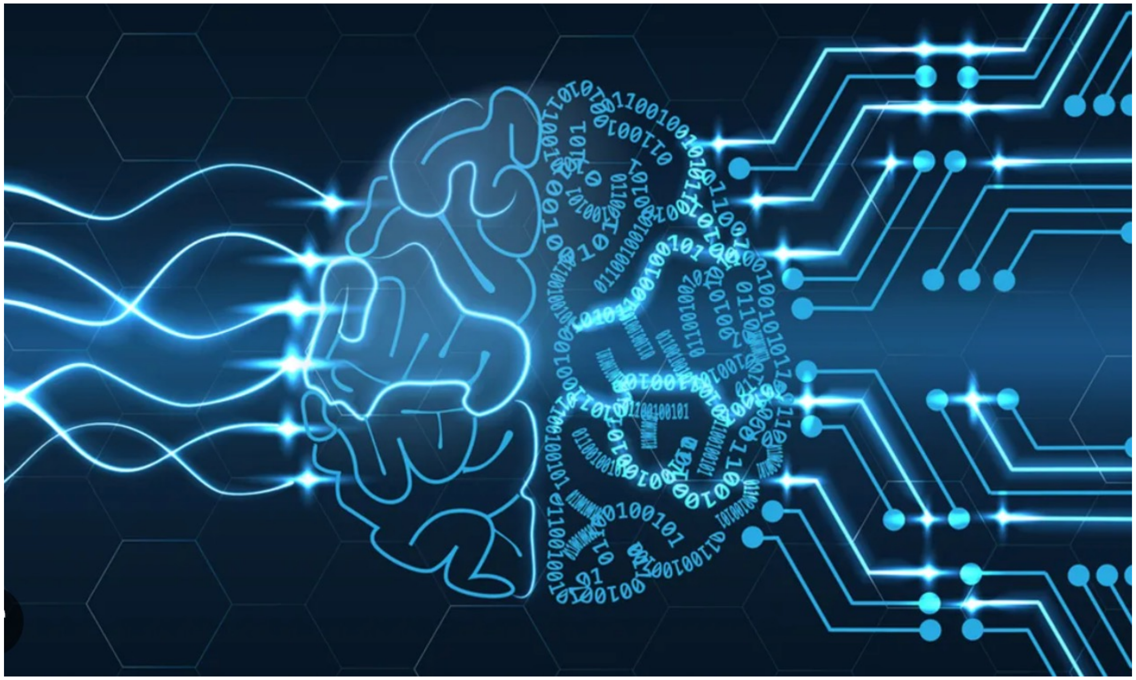
CS 4100: Introduction to AI

Wayne Snyder
Northeastern University

Lecture 20: Deep Learning – Text Classification Workflow



Deep Learning for Classification

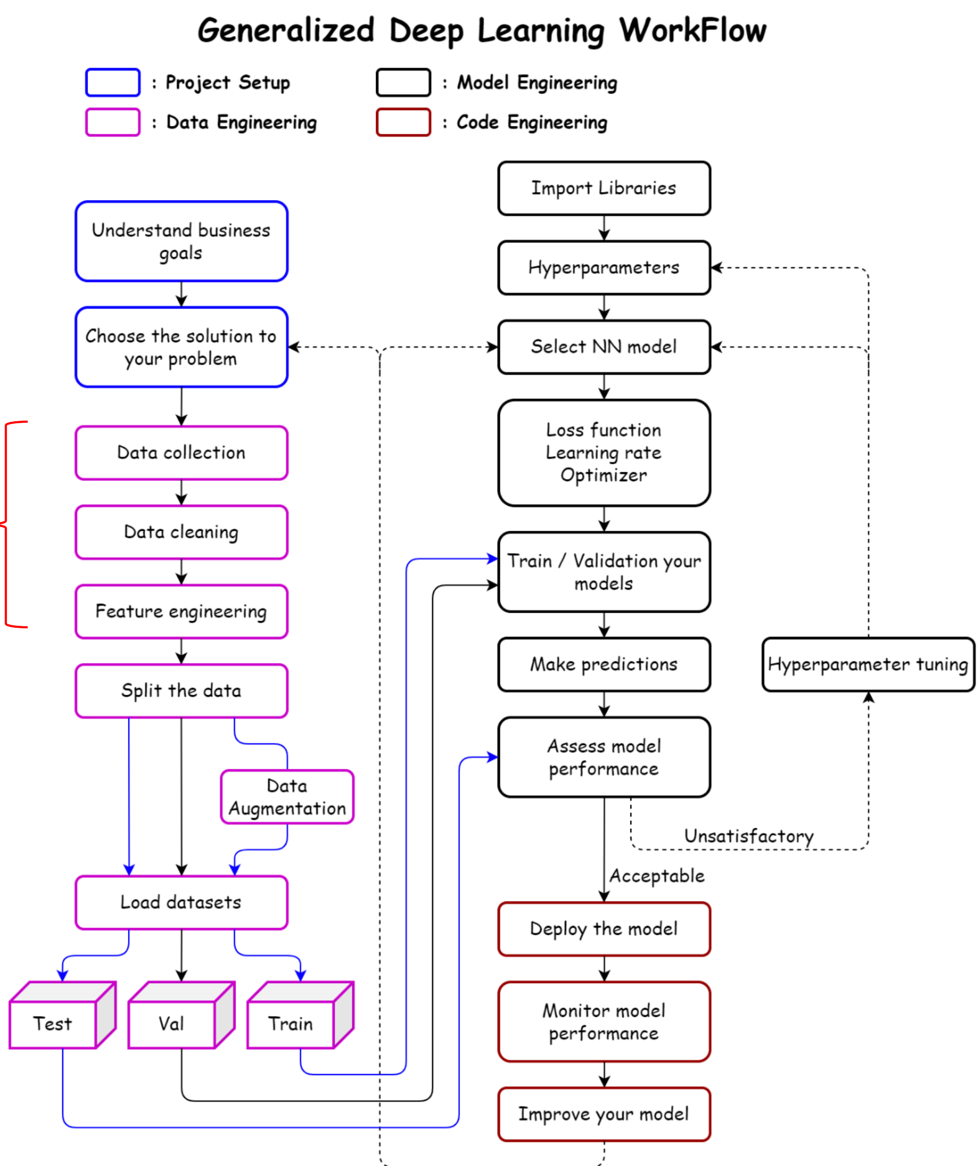
Plan for this Lecture:

- Supervised Learning for Classification
- Text Classification Workflow
- Example of Binary Classification: Positive and Negative Movie Reviews

Wednesday's Lecture:

- Multiple Classification of Handwritten Digits
- Convolution Networks for Image Processing

Deep Learning for Classification



Data Selection and Preparation

Data Formatting for Input to NN

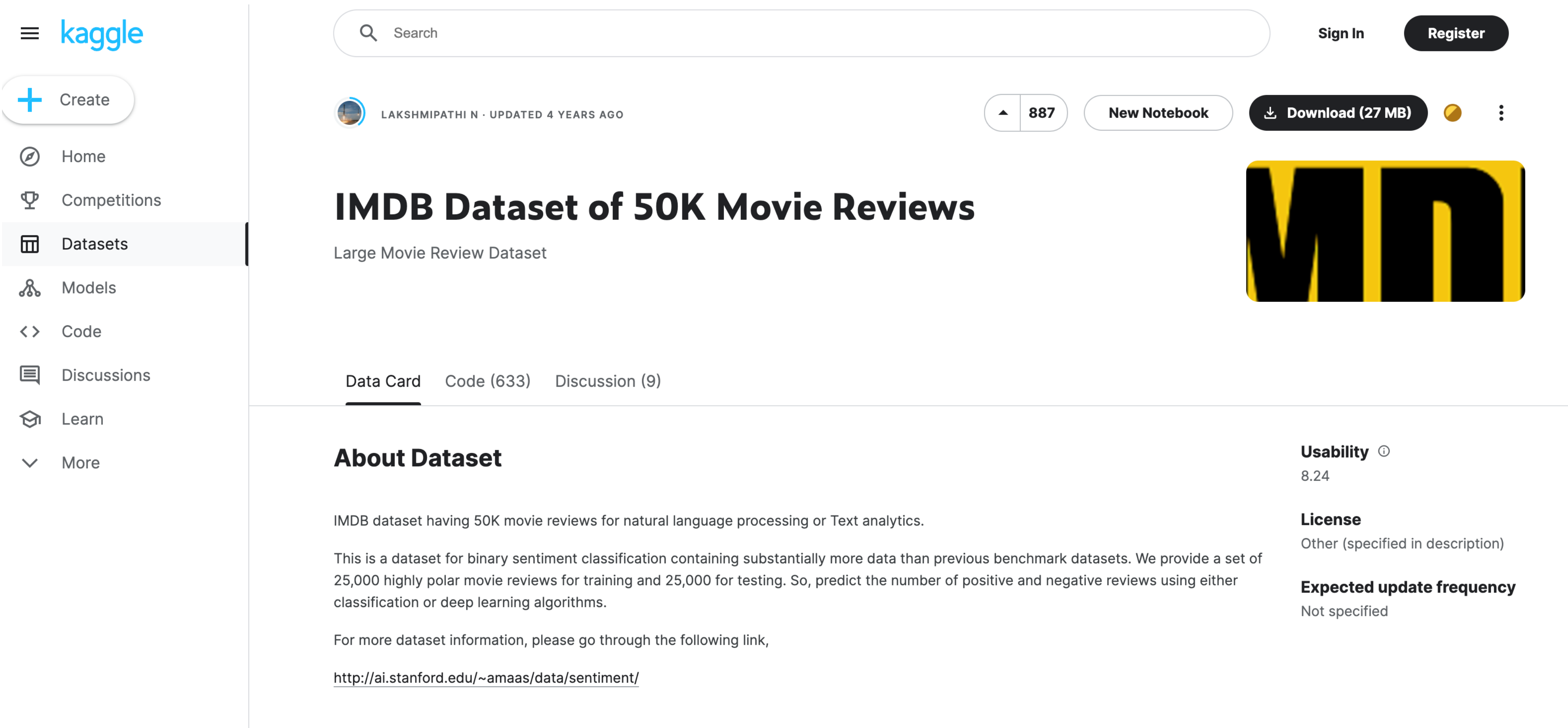
Deep Learning for Classification

Problem to be Solved: Classify movie reviews as "positive" or "negative"

Approach: Supervised Learning with Feedforward Neural Network

DL for Classification: Data Collection

There are many public data sets available, Kaggle is a good place to start!



DL for Classification: Data Collection

Data set consists of short texts with labels "positive" or "negative" (human annotated!).

In [2]:

```
#importing the training data
imdb_data=pd.read_csv('../input/IMDB Dataset.csv')
print(imdb_data.shape)
imdb_data.head(10)
```

```
(50000, 2)
```

Out[2]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
5	Probably my all-time favorite movie, a story o...	positive
6	I sure would like to see a resurrection of a u...	positive
7	This show was an amazing, fresh & innovative i...	negative
8	Encouraged by the positive comments about this...	negative
9	If you like original gut wrenching laughter yo...	positive

DL for Classification: Data Collection

It is often a good idea to do some preliminary exploration of your data to understand what you have:

Exploratery data analysis

In [3]:
```
#Summary of the dataset
imdb_data.describe()
```

Out[3]:

	review	sentiment
count	50000	50000
unique	49582	2
top	Loved today's show!!! It was a variety and not...	positive
freq	5	25000

Sentiment count

In [4]:
```
#sentiment count
imdb_data['sentiment'].value_counts()
```

Out[4]:
```
positive    25000
negative    25000
Name: sentiment, dtype: int64
```

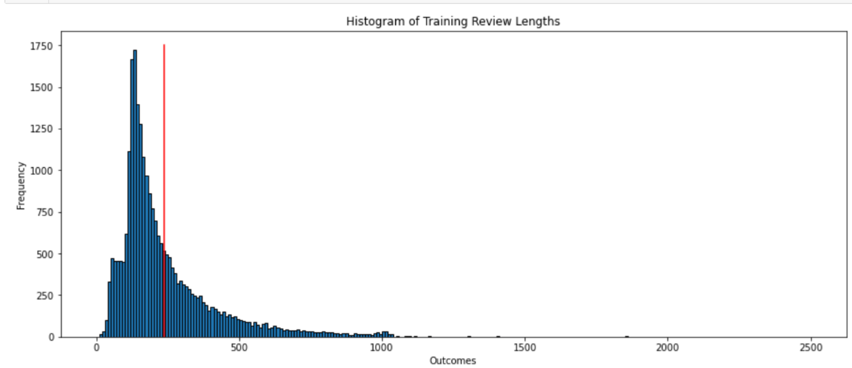
```
print("Length of reviews:")
for k in range(100):
    print(len(train_data[k]),' ', end='')
```

```
Length of reviews:
218  189  141  550  147  43  123  562  233  130  450  99  117  238  109  129  163  752
93  142  220  193  171  221  174  647  233  162  597  234  51  336  139  231  704  142
103  186  113  169  469  138  302  766  351  146  59  206  107  152  186  431  147  684
314  118  390  132  710  306  167  115  95  158  156  82  502  314  190  174  60  145
238  170  107  171
```

```
lens = [len(train_data[k]) for k in range(len(train_data))]
print("Shortest review:",min(lens))
print("Longest review:",max(lens))
print("Average length:",np.mean(lens))
```

```
Shortest review: 11
Longest review: 2494
Average length: 238.71364
```

4]:
```

m = np.mean(lens)
plt.figure(figsize=(15, 6))
plt.title("Histogram of Training Review Lengths")
plt.hist(lens,bins=np.arange(0,2501,10),edgecolor='black')
plt.plot([m,m],[0,1750],color='r')
plt.xlabel("Outcomes")
plt.ylabel("Frequency")
plt.show()
```



DL for Classification: Data Collection

It is often a good idea to do some preliminary exploration of your data to understand what you have:

Word cloud for positive review words

In [48]:

```
#word cloud for positive review words
plt.figure(figsize=(10,10))
positive_text=norm_train_reviews[1]
WC=WordCloud(width=1000,height=500,max_words=500,min_font_size=5)
positive_words=WC.generate(positive_text)
plt.imshow(positive_words,interpolation='bilinear')
plt.show
```

Out[48]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```

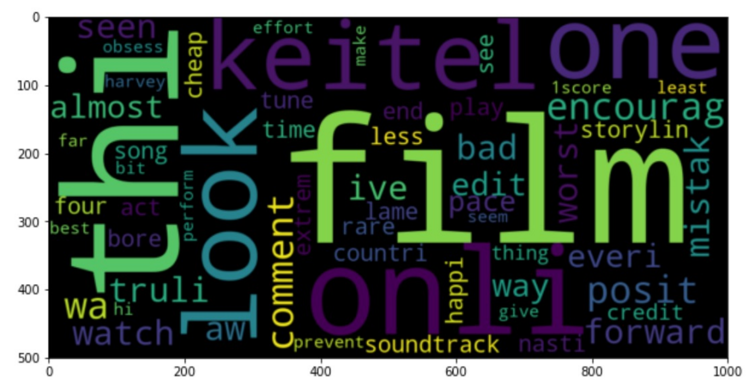
Word cloud for negative review words

In [49]:

```
#Word cloud for negative review words
plt.figure(figsize=(10,10))
negative_text=norm_train_reviews[8]
WC=WordCloud(width=1000,height=500,max_words=500,min_font_size=5)
negative_words=WC.generate(negative_text)
plt.imshow(negative_words,interpolation='bilinear')
plt.show
```

Out[49]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```



DL for Classification: Data Cleaning

Before we can input the text into the network, we need to do some cleaning (also called "normalization"):

Text normalization

```
In [6]:
#Tokenization of text
tokenizer=ToktokTokenizer()
#Setting English stopwords
stopword_list=nltk.corpus.stopwords.words('english')
```

Removing html strips and noise text

```
In [7]:
#Removing the html strips
def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

#Removing the square brackets
def remove_between_square_brackets(text):
    return re.sub('\[[^]]*\]', '', text)

#Removing the noisy text
def denoise_text(text):
    text = strip_html(text)
    text = remove_between_square_brackets(text)
    return text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(denoise_text)
```

Removing special characters

```
In [8]:
#Define function for removing special characters
def remove_special_characters(text, remove_digits=True):
    pattern=r'[^a-zA-z0-9\s]'
    text=re.sub(pattern,'',text)
    return text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(remove_special_characte
```

Text stemming

```
In [9]:
#Stemming the text
def simple_stemmer(text):
    ps=nltk.porter.PorterStemmer()
    text= ' '.join([ps.stem(word) for word in text.split()])
    return text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(simple_stemmer)
```

DL for Classification: Data Cleaning

Before we can input the text into the network, we need to do some cleaning (also called "normalization"):

Removing stopwords

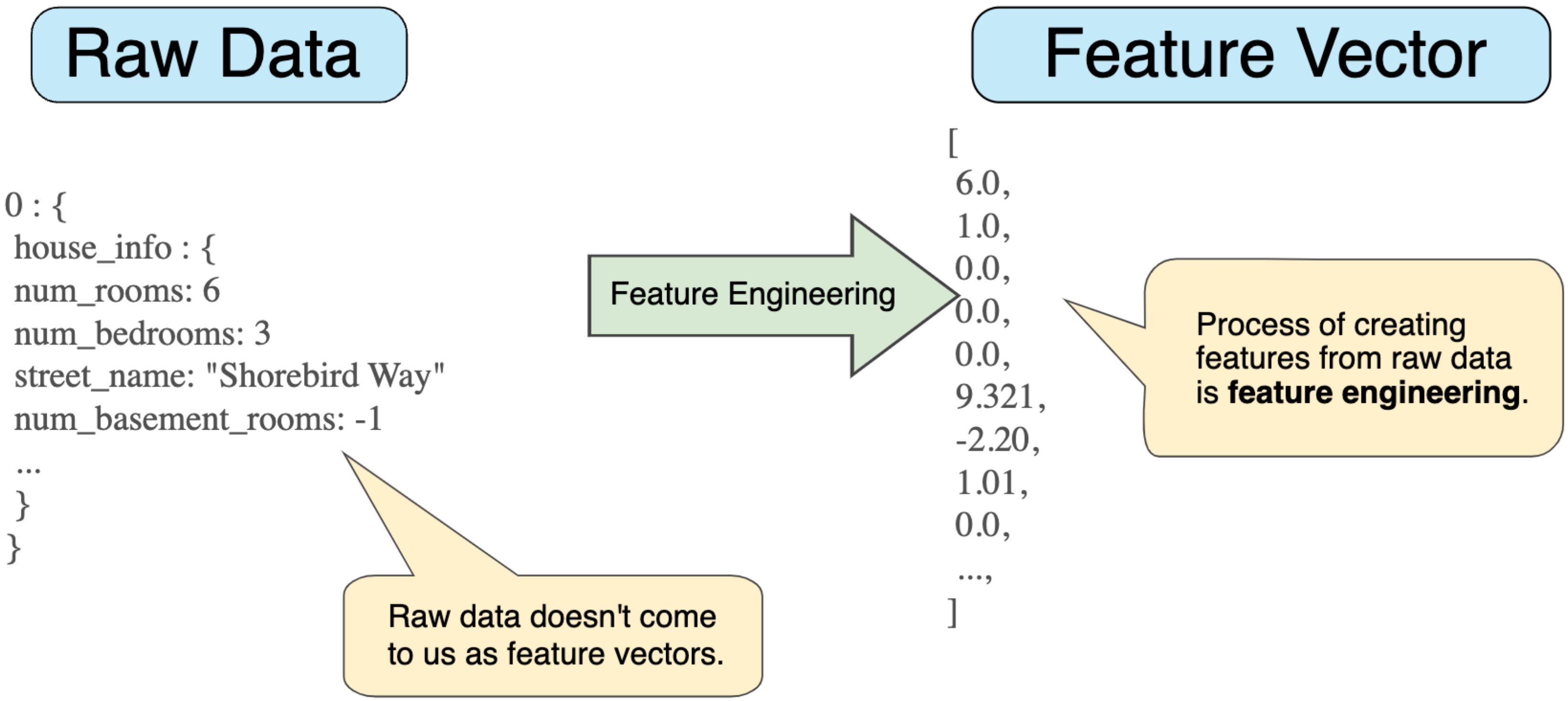
In [10]:

```
#set stopwords to english
stop=set(stopwords.words('english'))
print(stop)

#removing the stopwords
def remove_stopwords(text, is_lower_case=False):
    tokens = tokenizer.tokenize(text)
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in stopword_list]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not in stopword_list]
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(remove_stopwords)
```

```
{'those', 'she', 'should', 'you', 'won', "hadn't", 'will', 'whom', 'yourself', 'y', 'below', 'down', "shan't", 'we', 'your', 'of
f', 'haven', 'were', 'only', "doesn't", 'no', 'from', "couldn't", 'of', "hasn't", 'very', "she's", 'hers', 'hadn', 'wasn', 'ourse
lves', "won't", 'what', 'are', 'a', 'its', 'o', 'been', 'once', 'mustn', 'after', 'not', "that'll", 'this', 'by', 'shan', "should
n't", 'themselves', 't', 'was', 'me', 'herself', 'ma', "mightn't", 'shouldn', 'if', 'and', 're', "needn't", "you'll", 'these', 's
ame', 'as', 'with', 'mightn', 'don', 'hasn', 'their', 'so', 'being', 's', 'itself', 'theirs', 'own', 'did', 'do', 've', 'didn',
'there', 'few', "isn't", 'each', 'because', 'the', "wouldn't", 'for', 'am', "you've", 'needn', "it's", 'through', 'can', "shoul
d've", 'does', 'just', 'other', "don't", 'over', 'has', 'above', 'any', 'd', 'm', 'couldn', 'out', 'is', 'll', 'nor', 'doing', 'y
ourselves', "you'd", 'an', "wasn't", "weren't", 'during', 'doesn', 'under', 'who', 'to', "didn't", 'be', 'up', 'in', 'here', 'som
e', 'which', 'having', 'more', 'now', 'ain', 'while', 'that', 'on', 'then', 'her', 'how', 'such', 'when', 'all', 'too', 'before',
'have', 'my', 'than', 'i', 'into', 'yours', 'until', 'about', "you're", 'they', 'had', 'our', 'again', 'them', 'himself', 'ours',
'but', "haven't", "aren't", 'against', 'wouldn', 'at', "mustn't", 'between', 'where', 'both', 'him', 'aren', 'he', 'weren', 'or',
'why', 'it', 'further', 'most', 'myself', 'isn', 'his'}
```

DL for Classification: Feature Engineering

Feature Engineering: Encoding raw data (e.g., cleaned text) into numeric vectors suitable for input to neural network.



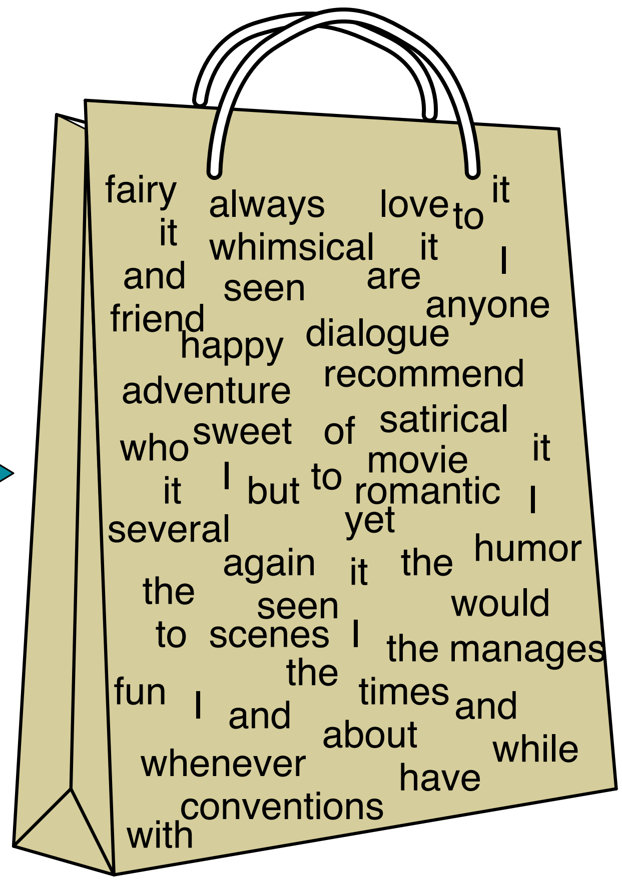
Feature values can be floats, integers, or bits 0/1.

DL for Classification: Feature Engineering

We will use a Bag-of-Words Encoding for Text:

- Create a vector of all words used in all reviews
- Each review is encoded as sparse vector of word counts
- Or: 0/1 indicating word is present or not ("Multi-Hot Encoding")

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
…	…

DL for Classification: Feature Engineering

Such encodings are useful throughout data science, and not just for text.

Label Encoding vs One-Hot Encoding

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

→

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

DL for Classification: Feature Engineering

Encoding the integer sequences via multi-hot encoding

In [9]:
```
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        for j in sequence:
            results[i, j] = 1.
    return results
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

In [101]:
```
x_train[0]
```

Out[101]:
```
array([0., 1., 1., ..., 0., 0., 0.])
```

In [97]:
```
y_train = np.asarray(train_labels).astype("float32")
y_test = np.asarray(test_labels).astype("float32")
y_test
```

Out[97]:
```
array([0., 1., 1., ..., 0., 0., 0.], dtype=float32)
```

In [102]:
```
train_labels
```

Out[102]:
```
array([1, 0, 0, ..., 0, 1, 0])
```

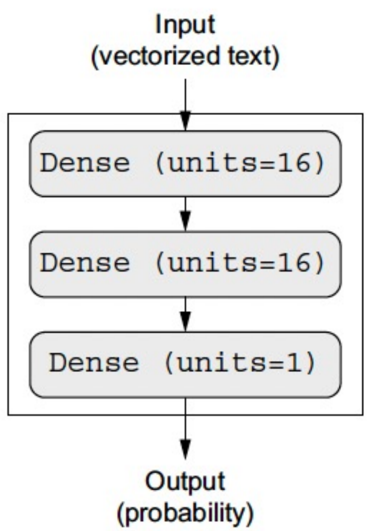
DL for Classification: Choosing a Model

Principal decisions to make about your model:

- How many layers (how deep)?
- What kind of layers and how wide?
- Output: binary (sigmoid), multiclass (softmax), etc...

In the beginning, one generally uses a model from a book, or a Google search; later you will gain experience about which architectures work best for which problems.

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```



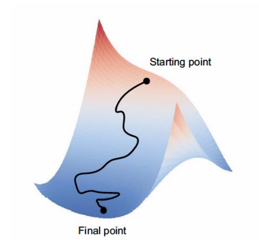
DL for Classification: Hyperparameters

Hyperparameters are choices you make about the algorithms used by the model:

```
Model.compile(
    optimizer="rmsprop",
    loss=None,
    metrics=None,
    loss_weights=None,
    weighted_metrics=None,
    run_eagerly=None,
    steps_per_execution=None,
    jit_compile=None,
    **kwargs
)
```

For most of these, you can just ignore them and use the defaults until you have more experience. The most important ones are:

- optimizer:
- loss
- metrics



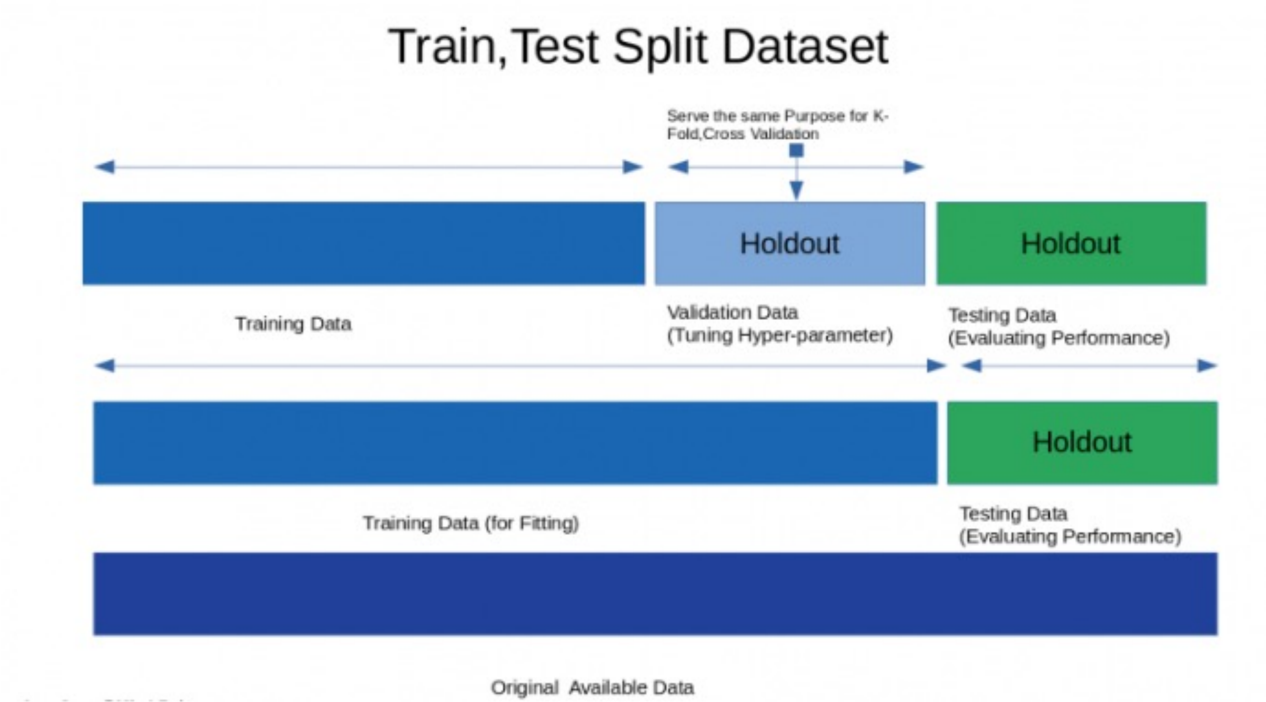

```
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

https://keras.io/api/models/model_training_apis/

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

DL for Classification: Training

In order to train the network so that it learns the best possible model of the data, it is necessary to "hold out" data separately from the training set, in order to validate during training, and test after training:



DL for Classification: Training

In the most sophisticated validation algorithm, the validation set is selected from different blocks of the training data and averaged:

K-fold cross-validation

DL for Classification: Training

You can split into explicit test and validation sets (Keras stores the IMDB example already split into training and testing sets. If you want Keras to do the validation split you can set the percentage to be held out during training:

```
n [52]:
# split into training and test data
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

```
In [3]: len(train_data)
Out[3]: 25000
```

```
In [4]: len(test_data)
Out[4]: 25000
```

Setting aside a validation set

```
In [17]:
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

```
In [18]: len(x_val)
Out[18]: 10000
```

```
In [19]: len(partial_x_train)
Out[19]: 15000
```

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    verbose='auto',
                    batch_size=512,
                    validation_split=0.2)
```

DL for Classification: Training

Training your model

In [20]:
```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

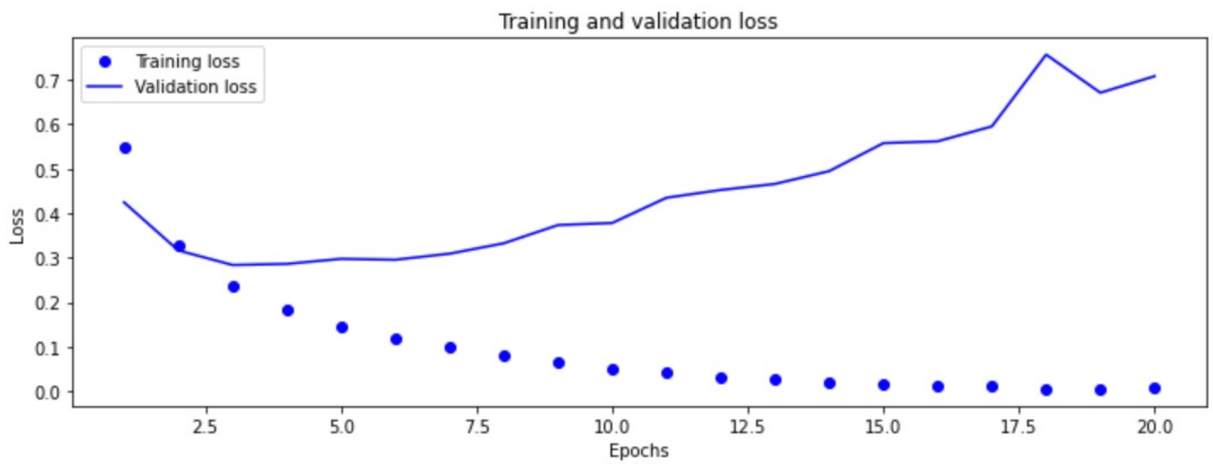
```
Epoch 1/20
30/30 [==============================] - 1s 33ms/step - loss: 0.6186 - accuracy: 0.6639 - val_loss: 0.4245 - val_accu
racy: 0.8552
Epoch 2/20
30/30 [==============================] - 0s 10ms/step - loss: 0.3477 - accuracy: 0.8966 - val_loss: 0.3164 - val_accu
racy: 0.8834
Epoch 3/20
30/30 [==============================] - 0s 11ms/step - loss: 0.2408 - accuracy: 0.9247 - val_loss: 0.2836 - val_accu
racy: 0.8897
Epoch 4/20
30/30 [==============================] - 0s 11ms/step - loss: 0.1819 - accuracy: 0.9476 - val_loss: 0.2860 - val_accu
racy: 0.8856
Epoch 5/20
30/30 [==============================] - 0s 11ms/step - loss: 0.1441 - accuracy: 0.9558 - val_loss: 0.2974 - val_accu
racy: 0.8799
Epoch 6/20
30/30 [==============================] - 0s 11ms/step - loss: 0.1234 - accuracy: 0.9635 - val_loss: 0.2954 - val_accu
racy: 0.8837
Epoch 7/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0960 - accuracy: 0.9723 - val_loss: 0.3090 - val_accu
racy: 0.8841
```

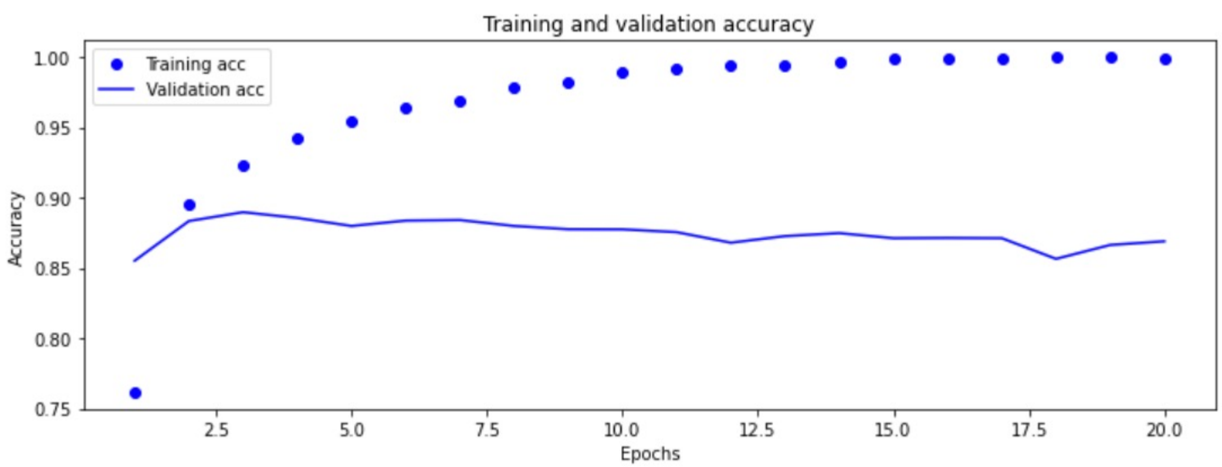
etc.....

```
Epoch 19/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0060 - accuracy: 0.9995 - val_loss: 0.6710 - val_accu
racy: 0.8664
Epoch 20/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0047 - accuracy: 0.9992 - val_loss: 0.7081 - val_accu
racy: 0.8690
```

DL for Classification: Training

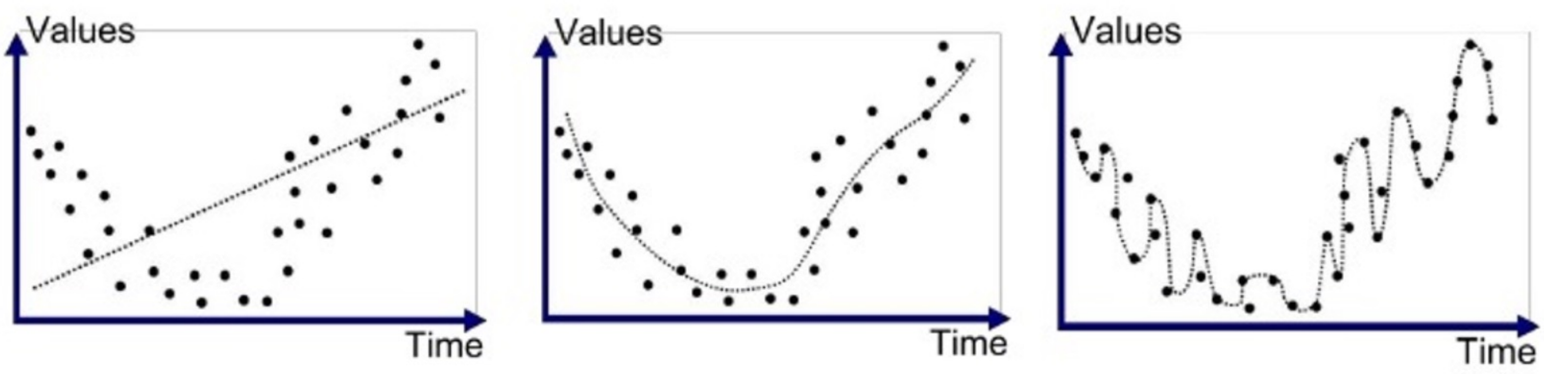
Plotting the training and validation loss and accuracy: What happened?





Deep Learning for Classification

One of the major issues you will need to confront is overfitting: the network has learned the training set too well – like a student who memorizes all the answers on a sample test, but never actually understood the problems!



Underfitted   Good Fit/Robust   Overfitted

Getting your network to generalize for a good fit is essential, or it won't work on new data!

DL for Classification: Training

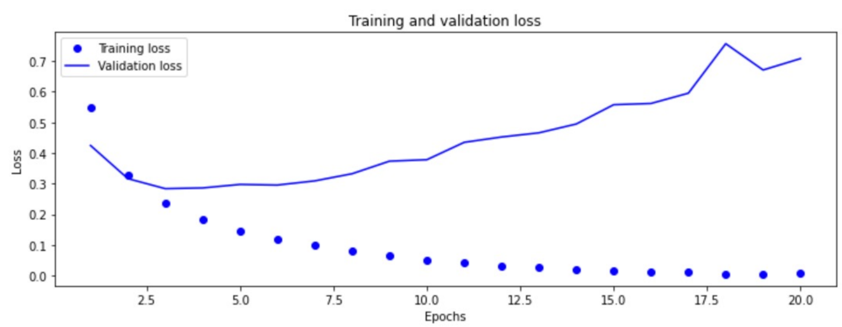
Finally, we test the model on the testing data. The accuracy is similar to the validation accuracy, which means that our validation strategy was appropriate:

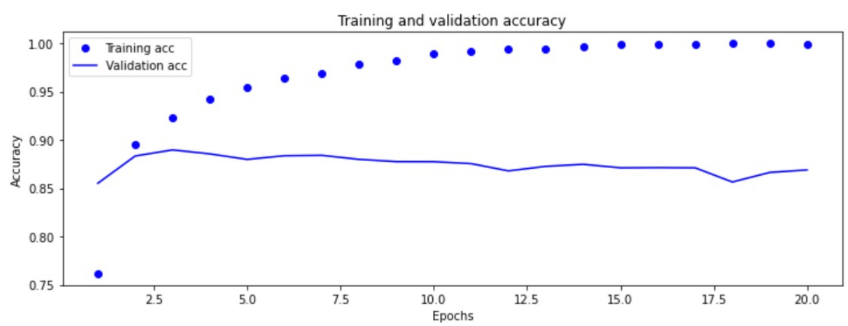
Testing your model

In [23]:
```
1 model.evaluate(x_test, y_test)
```

```
782/782 [==============================] - 1s 752us/step - loss: 0.8323 - accuracy: 0.8407
```

Out[23]:
```
[0.8322579860687256, 0.8406800031661987]
```





Deep Learning for Classification

A naive but often useful solution is early stopping: stop the training before it starts to overfit. In our case it seems like 4 epochs would be better:

Retraining a model from scratch

In [23]:
```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 [==============================] - 1s 8ms/step - loss: 0.5300 - accuracy: 0.7549
Epoch 2/4
49/49 [==============================] - 0s 8ms/step - loss: 0.2674 - accuracy: 0.9079
Epoch 3/4
49/49 [==============================] - 0s 8ms/step - loss: 0.1955 - accuracy: 0.9339
Epoch 4/4
49/49 [==============================] - 0s 8ms/step - loss: 0.1586 - accuracy: 0.9436
782/782 [==============================] - 1s 732us/step - loss: 0.2965 - accuracy: 0.8832
```

In [24]:
```
results
```

Out[24]:
```
[0.2965046167373657, 0.8831599950790405]
```

Deep Learning for Classification

We can now use our model to predict the sentiment for a new piece of data (we'll just show it on the testing examples):

Using a trained model to generate predictions on new data

In [26]:
```
model.predict(x_test)
```

Out[26]:
```
array([[0.2766113 ],
       [0.99981654],
       [0.90646017],
       ...,
       [0.17581889],
       [0.10580322],
       [0.7304399 ]], dtype=float32)
```

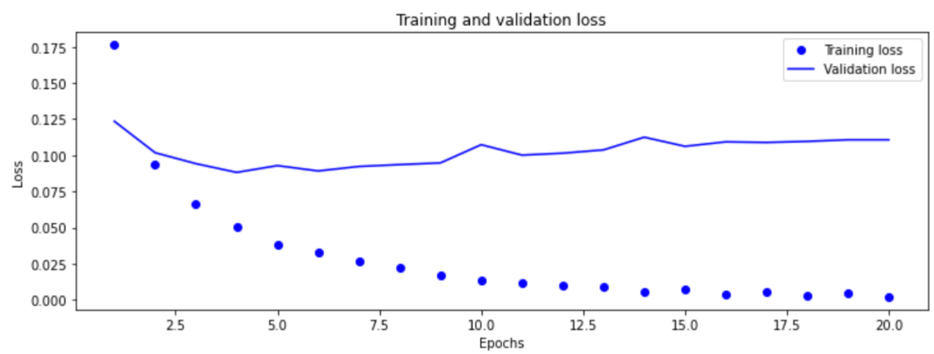
Deep Learning for Classification

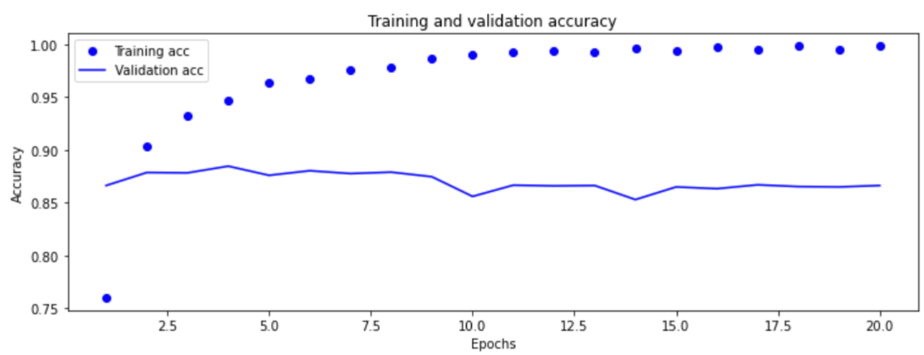
What next? How to improve our results?

- Try different data preparation (e.g., removing stop words or not)
- Get MORE DATA
- Try a different architecture.
- Try different hyperparameters.

Deep Learning for Classification

In [34]:
```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model.compile(optimizer="rmsprop",
              loss="mse",                # mean square error
              metrics=["accuracy"])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    verbose=0,
                    batch_size=512,
                    validation_split=0.2)

display_graphs(history)

model.evaluate(x_test, y_test)
```



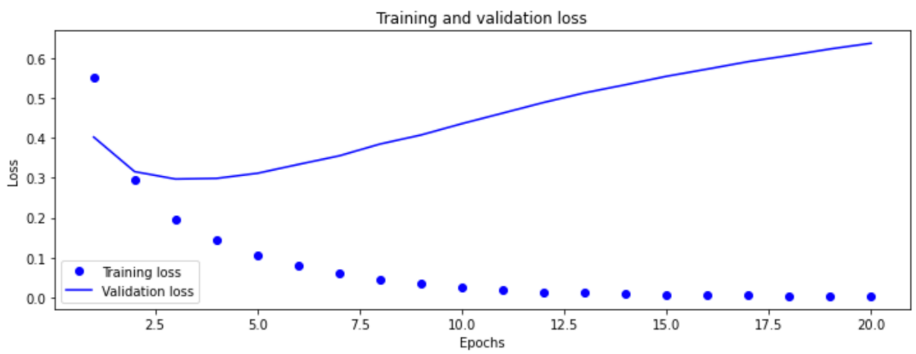


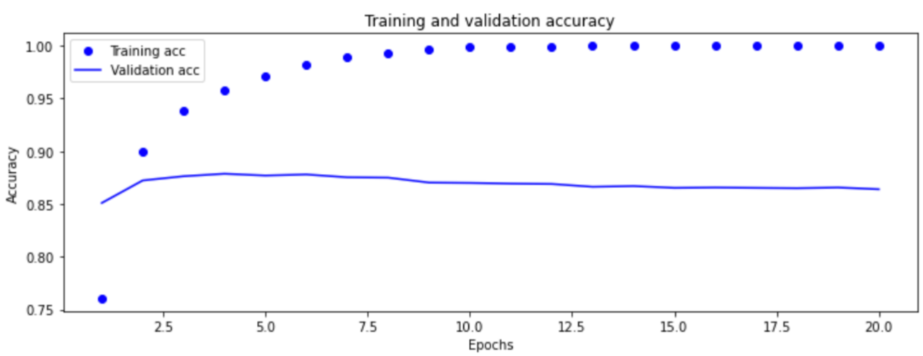

```
782/782 [==============================] - 1s 772us/step - loss: 0.1210 - accuracy: 0.8489
```

Out[34]: `[0.12096309661865234, 0.8488799929618835]`

Deep Learning for Classification

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam",
              loss="binary_crossentropy",
              metrics=["accuracy"])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    verbose=0,
                    batch_size=512,
                    validation_split=0.2)

display_graphs(history)

model.evaluate(x_test, y_test)
```

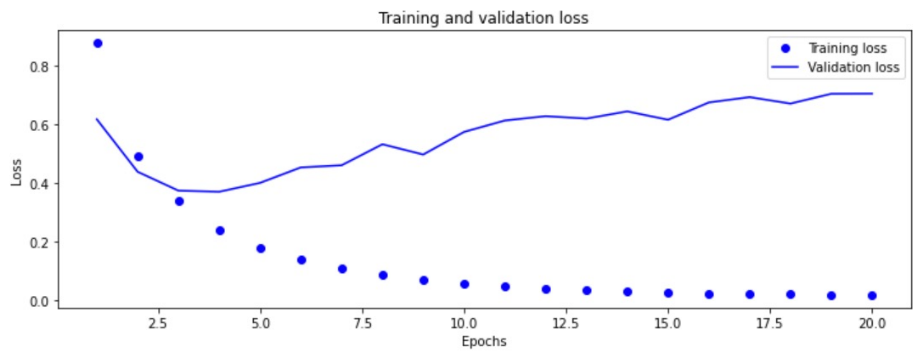


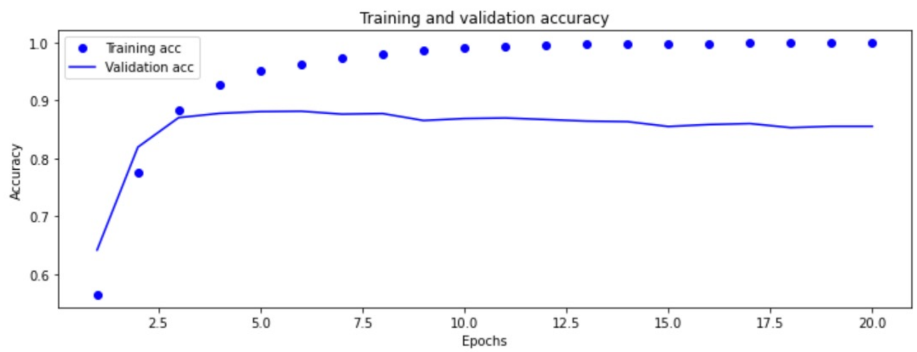



```
782/782 [==============================] - 1s 772us/step - loss: 0.6593 - accuracy: 0.8552
[0.659318208694458, 0.855239987373352]
```

Deep Learning for Classification

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="tanh")
])

model.compile(optimizer="adam",
              loss="binary_crossentropy",
              metrics=["accuracy"])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    verbose=0,
                    batch_size=512,
                    validation_split=0.2)

display_graphs(history)

model.evaluate(x_test, y_test)
```

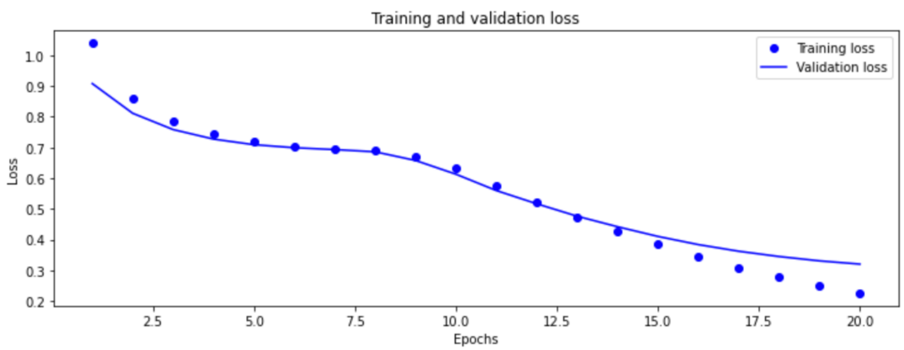


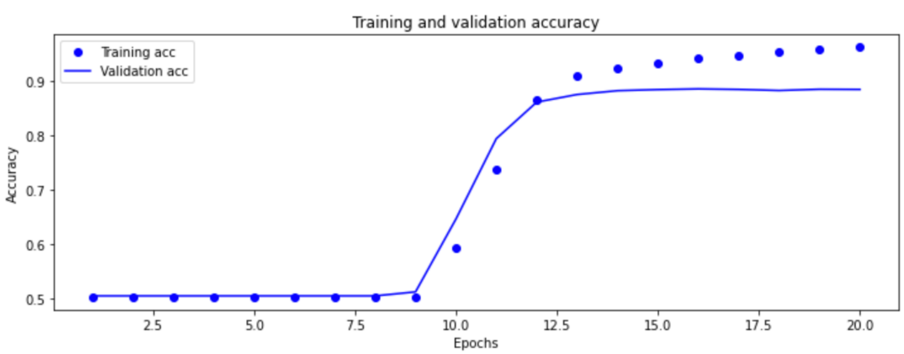



```
782/782 [==============================] - 1s 790us/step - loss: 0.7652 - accuracy: 0.8544

[0.7651742696762085, 0.8543599843978882]
```

Deep Learning for Classification

```
model = keras.Sequential([
    layers.Dense(16, activation="sigmoid"),
    layers.Dense(16, activation="sigmoid"),
    layers.Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam",
              loss="binary_crossentropy",
              metrics=["accuracy"])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    verbose=0,
                    batch_size=512,
                    validation_split=0.2)

display_graphs(history)

model.evaluate(x_test, y_test)
```



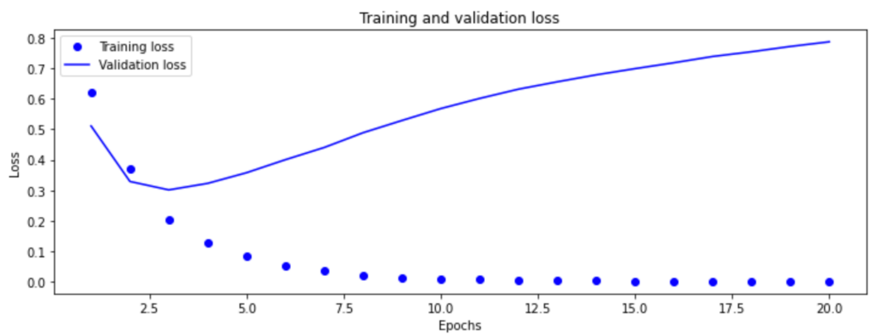


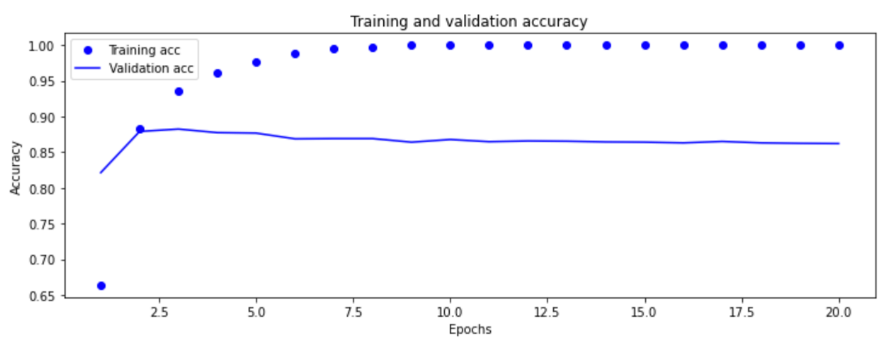

```
782/782 [==============================] - 1s 781us/step - loss: 0.3267 - accuracy: 0.8849
```

```
[0.3267013430595398, 0.8849200010299683]
```

Deep Learning for Classification

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam",
              loss="binary_crossentropy",
              metrics=["accuracy"])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    verbose=0,
                    batch_size=512,
                    validation_split=0.2)

display_graphs(history)

model.evaluate(x_test, y_test)
```

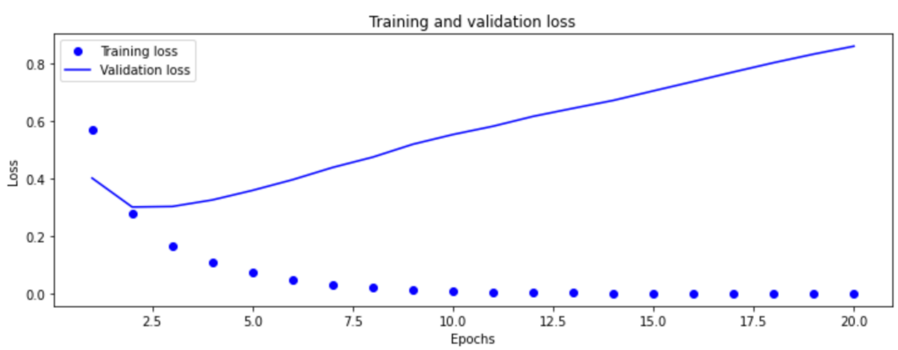


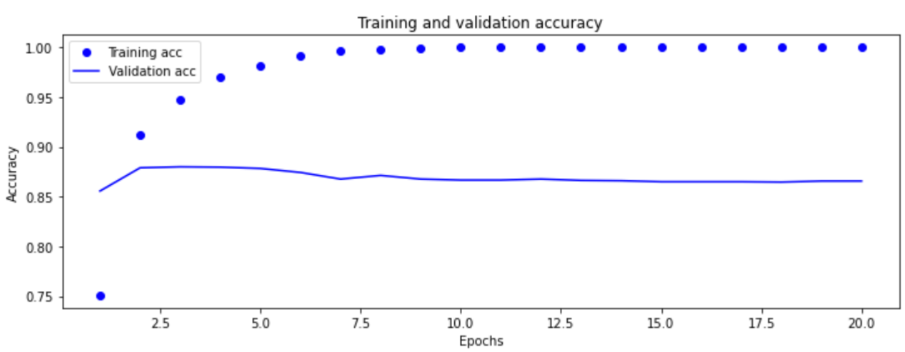



```
782/782 [==============================] - 1s 790us/step - loss: 0.8209 - accuracy: 0.8547
[0.8208646774291992, 0.8547199964523315]
```

Deep Learning for Classification

```python
model = keras.Sequential([
    layers.Dense(32, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam",
              loss="binary_crossentropy",
              metrics=["accuracy"])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    verbose=0,
                    batch_size=512,
                    validation_split=0.2)

display_graphs(history)

model.evaluate(x_test, y_test)
```

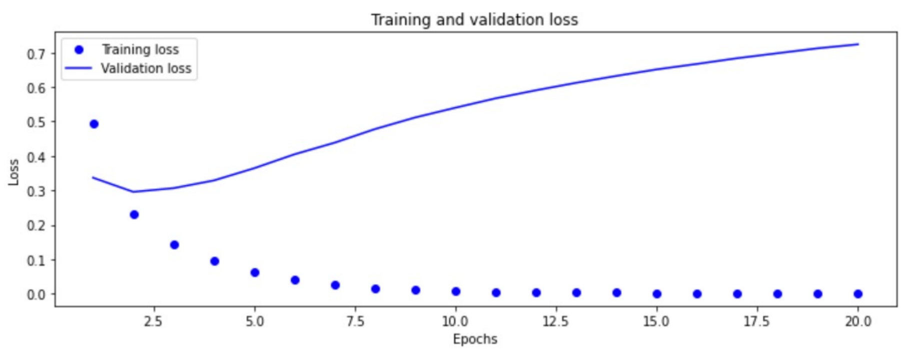


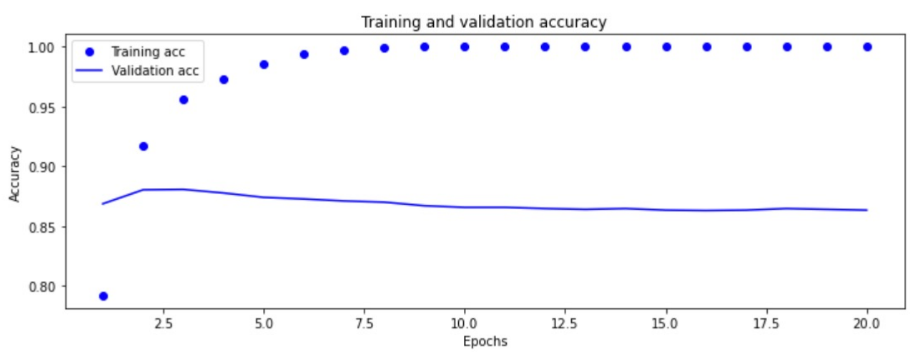



```
782/782 [==============================] - 1s 814us/step - loss: 0.8836 - accuracy: 0.8555
[0.8836396336555481, 0.8555200099945068]
```

Deep Learning for Classification

```python
model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam",
              loss="binary_crossentropy",
              metrics=["accuracy"])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    verbose=0,
                    batch_size=512,
                    validation_split=0.2)

display_graphs(history)

model.evaluate(x_test, y_test)
```



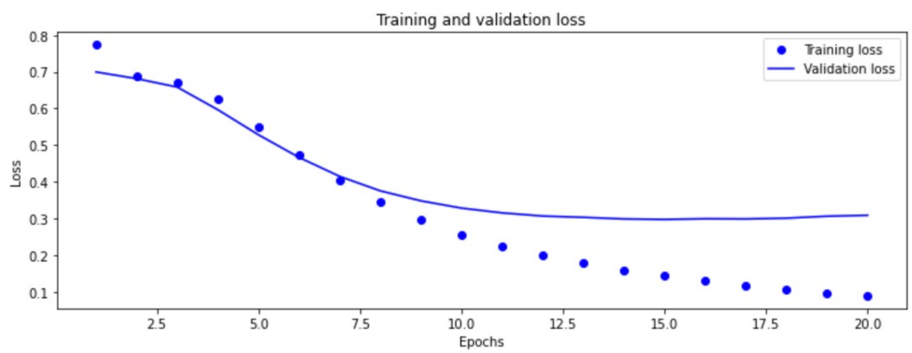


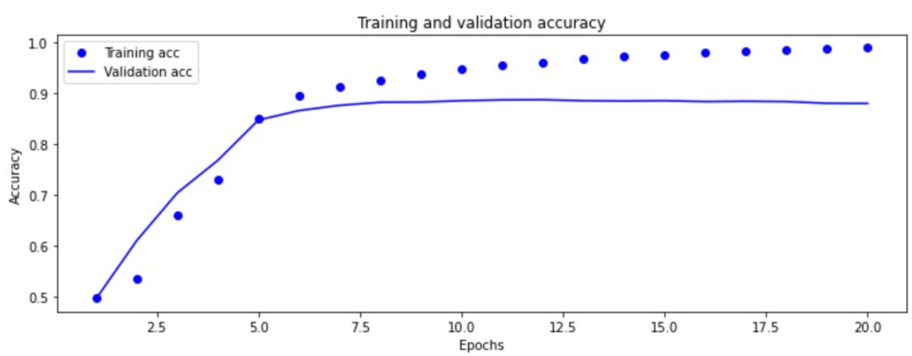

```
782/782 [==============================] - 1s 812us/step - loss: 0.7531 - accuracy: 0.8550
```

```
[0.7530536651611328, 0.8550400137901306]
```

Deep Learning for Classification

```
model = keras.Sequential([
    layers.Dense(32, activation="sigmoid"),
    layers.Dense(16, activation="sigmoid"),
    layers.Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam",
              loss="binary_crossentropy",
              metrics=["accuracy"])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    verbose=0,
                    batch_size=512,
                    validation_split=0.2)

display_graphs(history)

model.evaluate(x_test, y_test)
```



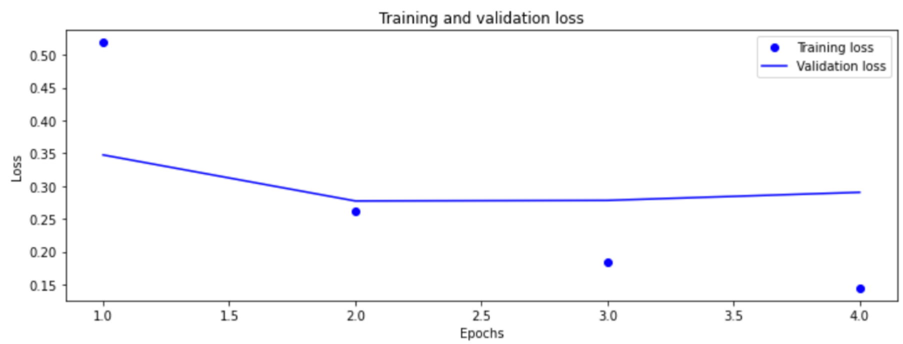


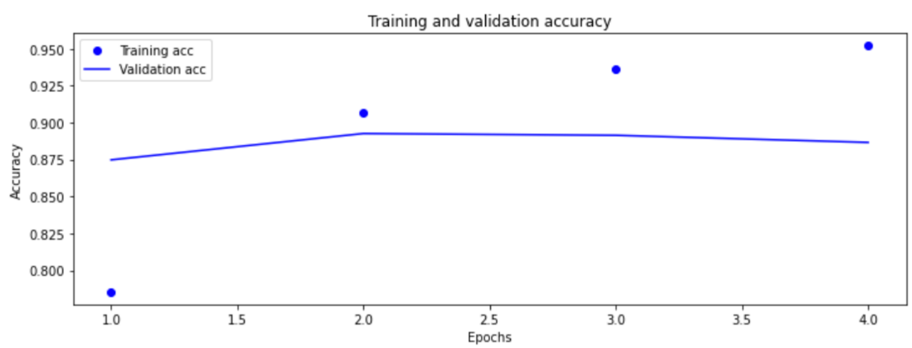

```
782/782 [==============================] - 1s 809us/step - loss: 0.3176 - accuracy: 0.8739
```

```
[0.31761646270751953, 0.8738800287246704]
```

Deep Learning for Classification

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam",
              loss="binary_crossentropy",
              metrics=["accuracy"])

history = model.fit(x_train,
                    y_train,
                    epochs=4,
                    verbose=0,
                    batch_size=512,
                    validation_split=0.2)

display_graphs(history)

model.evaluate(x_test, y_test)
```






```
782/782 [==============================] - 1s 834us/step - loss: 0.3091 - accuracy: 0.8791

[0.3090904653072357, 0.8791199922561646]
```