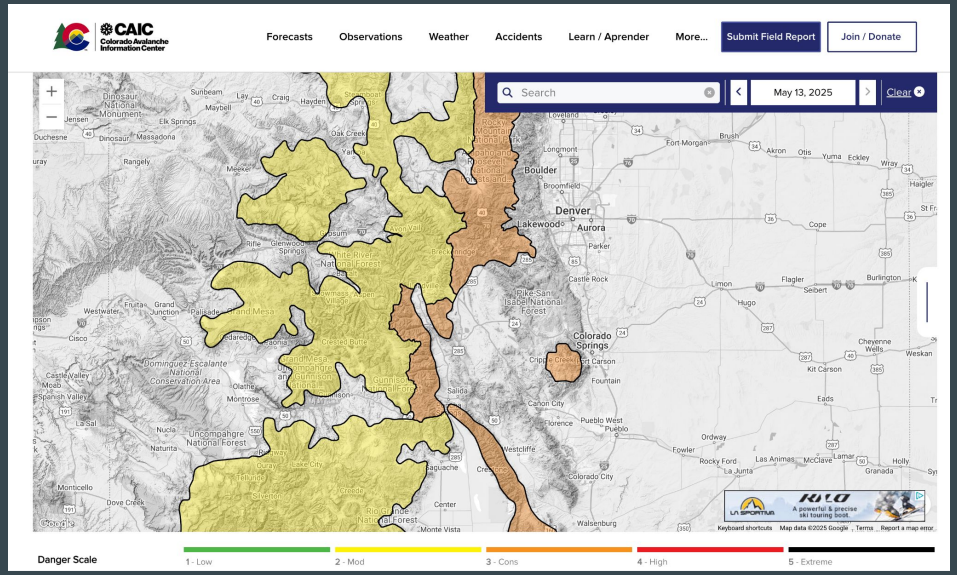
Avalanche Forecast Deep Learning

Sam Vredenburgh

Introduction

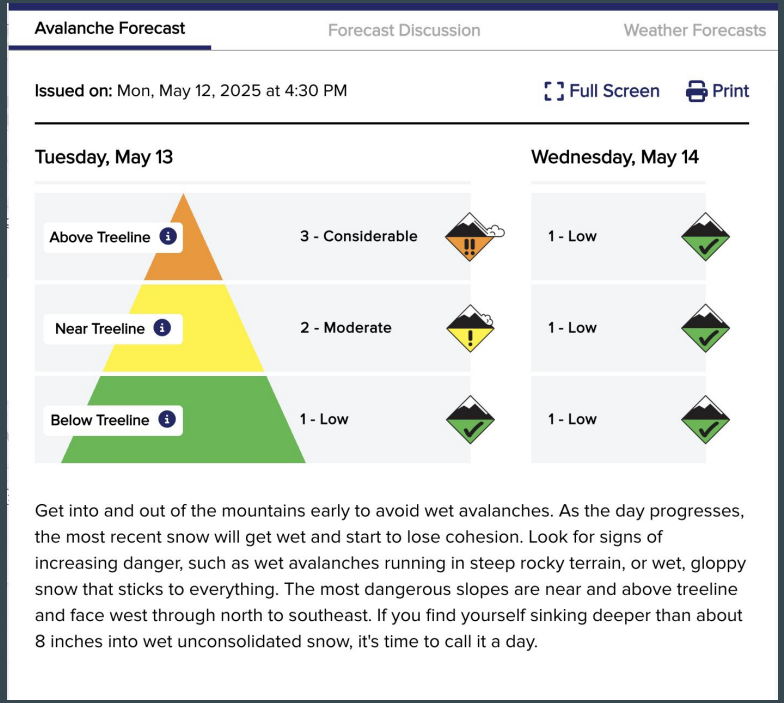
As an avid backcountry skier, the Colorado Avalanche Information Center (CAIC) has been an invaluable resource, providing avalanche danger forecasts across the state at all aspects and elevations. For my final project I wanted to incorporate the text forecast summaries that appear on the front page of each forecast zone, along with the forecasted danger levels at each elevation tier corresponding to the forecast summary text.



Setup

The text is typically 1-2 paragraphs, the elevation tiers are always **Below Treeline**, **Near Treeline** and **Above Treeline**, and the danger level for each elevation tier is a categorical scale as follows:

- **0 - earlySeason**
- **1 - Low**
- **2 - Moderate**
- **3 - Considerable**
- **4 - High**
- **5 - Extreme**



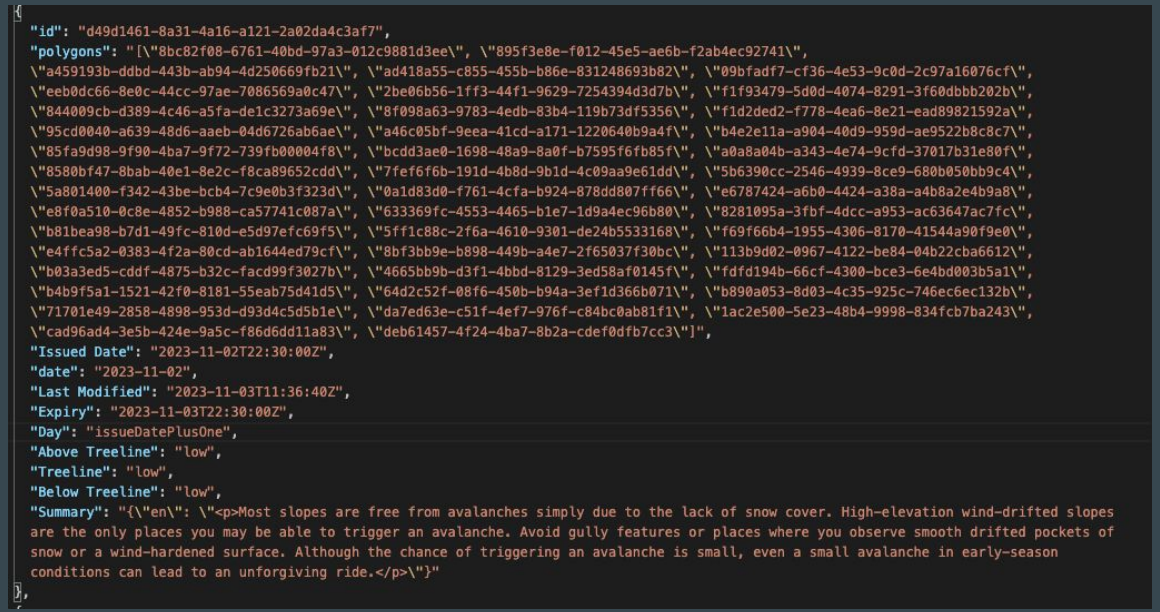
Goal

The goal is to build a multi-class multi-output classification model to predict the danger level at each elevation tier, given the input summary text. I decided to use multiple text pre-processing techniques and model architectures including:

- Bag of Words
- Bigrams
- TF-IDF
- Bidirectional LSTM
- Word Embeddings
- TransformerEncoder
- Pre-trained Transformer

Data Acquisition

The first step was to reach out to the CAIC and request historical forecast data. After initial enthusiasm, it took over a month of back and forth to get a json file of past forecast summaries and associated danger levels per elevation sent over.



```
{
  "id": "d49d1461-8a31-4a16-a121-2a02da4c3af7",
  "polygons": "[\"8bc82f08-6761-40bd-97a3-012c9881d3ee\", \"895f3e8e-f012-45e5-ae6b-f2ab4ec92741\",
  \"a459193b-ddbd-443b-ab94-4d250669fb21\", \"ad418a55-c855-455b-b86e-831248693b82\", \"09bfadf7-cf36-4e53-9c0d-2c97a16076cf\",
  \"eeb0dc66-8e0c-44cc-97ae-7086569a0c47\", \"2be06b56-1ff3-44f1-9629-7254394d3d7b\", \"f1f93479-5d0d-4074-8291-3f60dbbb202b\",
  \"844009cb-d389-4c46-a5fa-de1c3273a69e\", \"8f098a63-9783-4edb-83b4-119b73df5356\", \"f1d2ded2-f778-4ea6-8e21-ead89821592a\",
  \"95cd0040-a639-48d6-aaeb-04d6726ab6ae\", \"a46c05bf-9eea-41cd-a171-1220640b9a4f\", \"b4e2e11a-a904-40d9-959d-ae9522b8c8c7\",
  \"85fa9d98-9f90-4ba7-9f72-739fb00004f8\", \"bcdd3ae0-1698-48a9-8a0f-b7595f6fb85f\", \"a0a8a04b-a343-4e74-9cfd-37017b31e80f\",
  \"8580bf47-8bab-40e1-8e2c-f8ca89652cdd\", \"7fef6f6b-191d-4b8d-9b1d-4c09aa9e61dd\", \"5b6390cc-2546-4939-8ce9-680b050bb9c4\",
  \"5a801400-f342-43be-bcb4-7c9e0b3f323d\", \"0a1d83d0-f761-4cfa-b924-878dd807ff66\", \"e6787424-a6b0-4424-a38a-a4b8a2e4b9a8\",
  \"e8f0a510-0c8e-4852-b988-ca57741c087a\", \"633369fc-4553-4465-b1e7-1d9a4ec96b80\", \"8281095a-3fbf-4dcc-a953-ac63647ac7fc\",
  \"b81bea98-b7d1-49fc-810d-e5d97efc69f5\", \"5ff1c88c-2f6a-4610-9301-de24b5533168\", \"f69f66b4-1955-4306-8170-41544a90f9e0\",
  \"e4ffc5a2-0383-4f2a-80cd-ab1644ed79cf\", \"8bf3bb9e-b898-449b-a4e7-2f65037f30bc\", \"113b9d02-0967-4122-be84-04b22cba6612\",
  \"b03a3ed5-cddf-4875-b32c-facd99f3027b\", \"4665bb9b-d3f1-4bbd-8129-3ed58af0145f\", \"fdfd194b-66cf-4300-bce3-6e4bd003b5a1\",
  \"b4b9f5a1-1521-42f0-8181-55eab75d41d5\", \"64d2c52f-08f6-450b-b94a-3ef1d366b071\", \"b890a053-8d03-4c35-925c-746ec6ec132b\",
  \"71701e49-2858-4898-953d-d93d4c5d5b1e\", \"da7ed63e-c51f-4ef7-976f-c84bc0ab81f1\", \"1ac2e500-5e23-48b4-9998-834fcb7ba243\",
  \"cad96ad4-3e5b-424e-9a5c-f86d6dd11a83\", \"deb61457-4f24-4ba7-8b2a-cdef0dfb7cc3\"]",
  "Issued Date": "2023-11-02T22:30:00Z",
  "date": "2023-11-02",
  "Last Modified": "2023-11-03T11:36:40Z",
  "Expiry": "2023-11-03T22:30:00Z",
  "Day": "issueDatePlusOne",
  "Above Treeline": "low",
  "Treeline": "low",
  "Below Treeline": "low",
  "Summary": "{\"en\": \"<p>Most slopes are free from avalanches simply due to the lack of snow cover. High-elevation wind-drifted slopes
  are the only places you may be able to trigger an avalanche. Avoid gully features or places where you observe smooth drifted pockets of
  snow or a wind-hardened surface. Although the chance of triggering an avalanche is small, even a small avalanche in early-season
  conditions can lead to an unforgiving ride.</p>\"}"
},
```

Data Wrangling

The initial json file had the raw forecast summary in unparsed html and had json errors stemming from links, <a> tags, that caused issues with the hrefs having double quotes. The initial raw data file contained 3334 forecasts. This was then whittled down to 2885 after creating a dataframe and removing ‘noRating’ instances and empty forecast texts. I later manually pulled another 56 summaries and danger levels from the CAIC website, and after concatenating those with the original data was left with a total of 2941 instances.

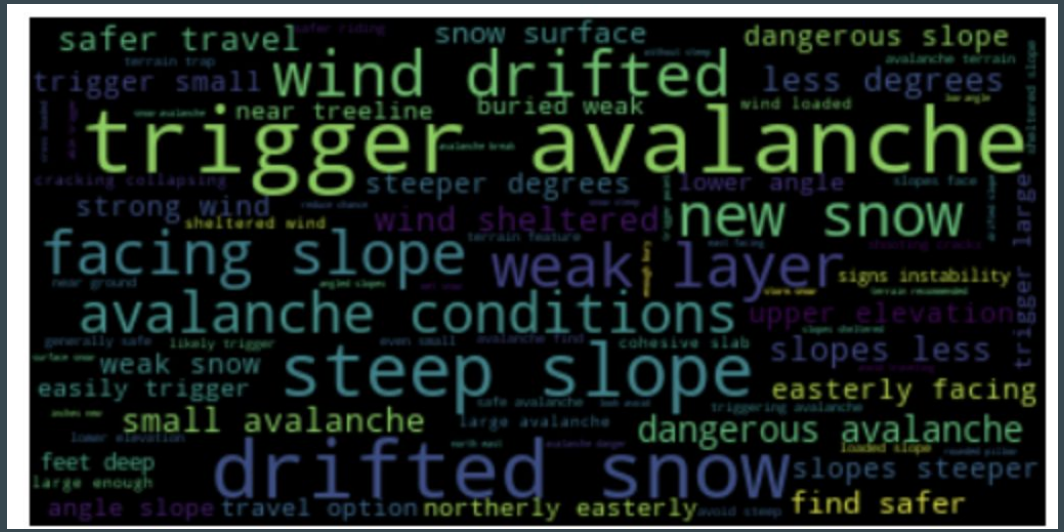
EDA - Labels

Upon inspecting the data, I found there were 0 instances of the danger level 5, ‘Extreme’, but some instances of ‘earlySeason’ so I decided to map the output class labels as:

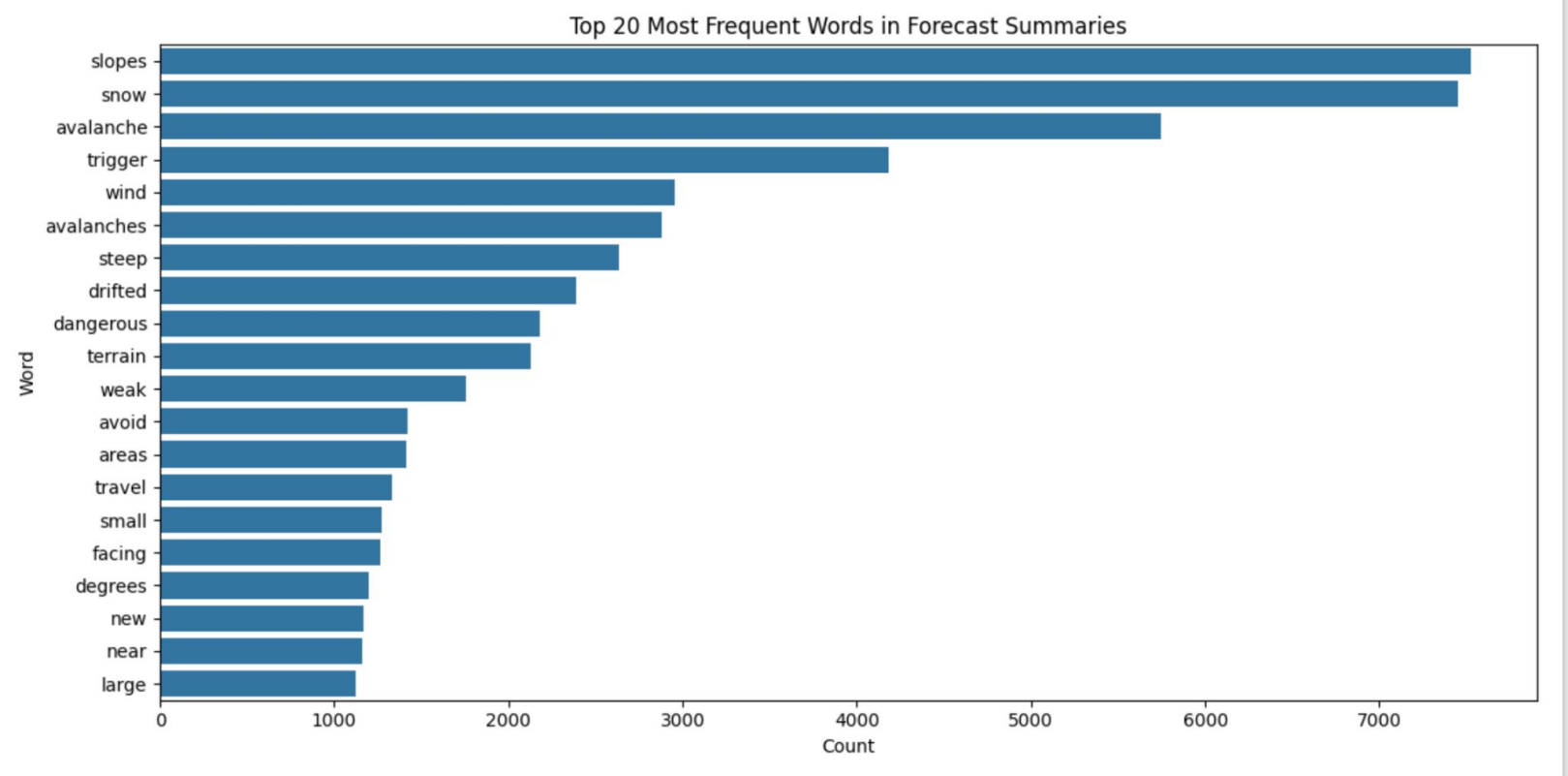
- 0 - earlySeason
- 1 - low
- 2 - moderate
- 3 - considerable
- 4 - high

EDA - Text

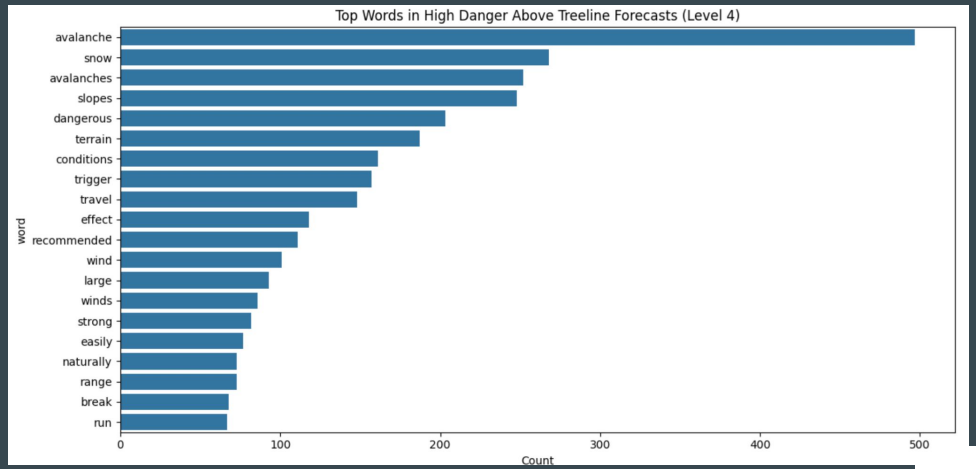
I used the ‘nltk’ package ‘stopwords’ to create a new column with the stopwords and punctuation removed, and all words to lowercase. From there I performed some visual EDA including a wordcloud visual, different plots for the most used words across all summaries, and corresponding to the different danger levels. I created a histogram of the distribution of summary lengths, and finally performed a TF-IDF analysis to discover the most informative words across all of the summaries.

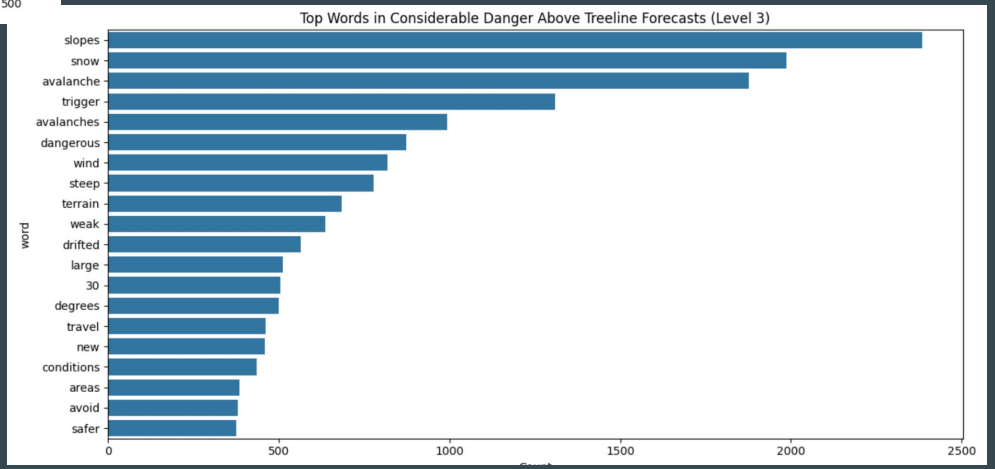


EDA - Top Words Overall

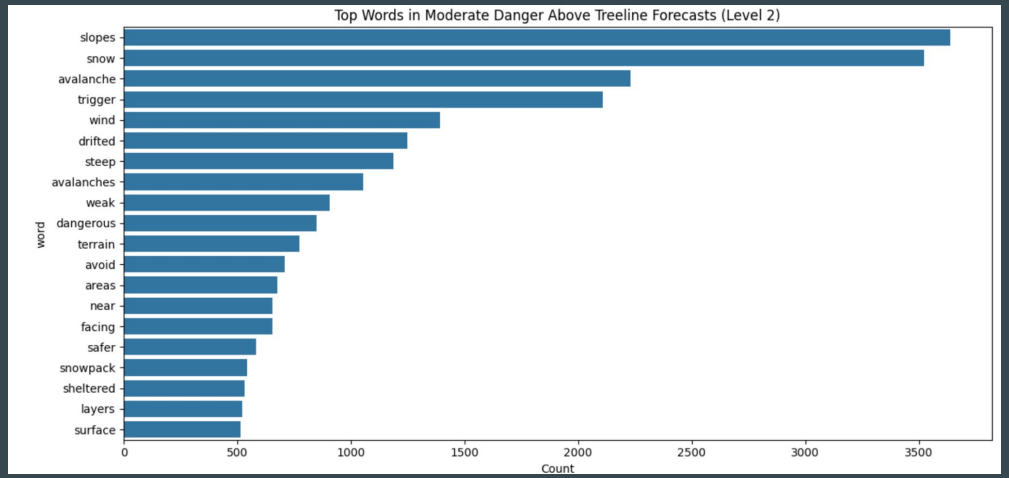


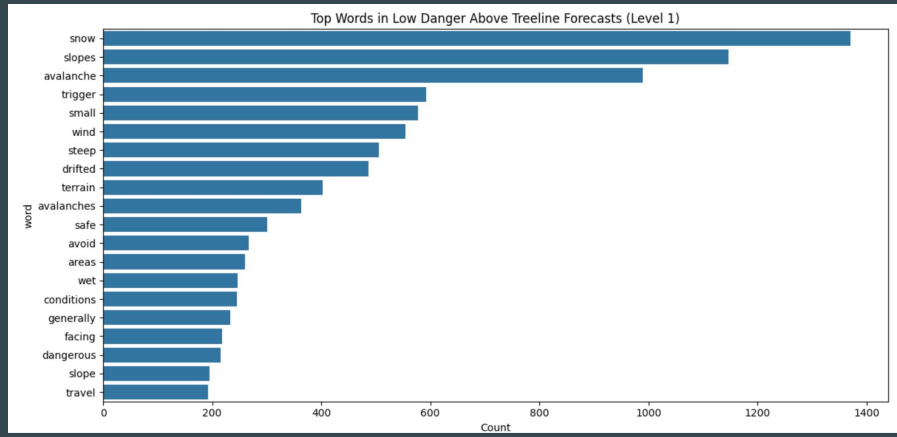
EDA - Top Words per Output (Above Treeline Only)

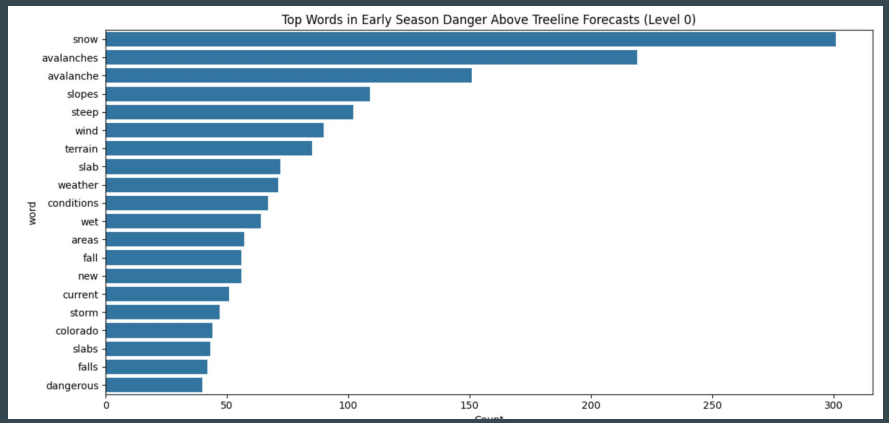




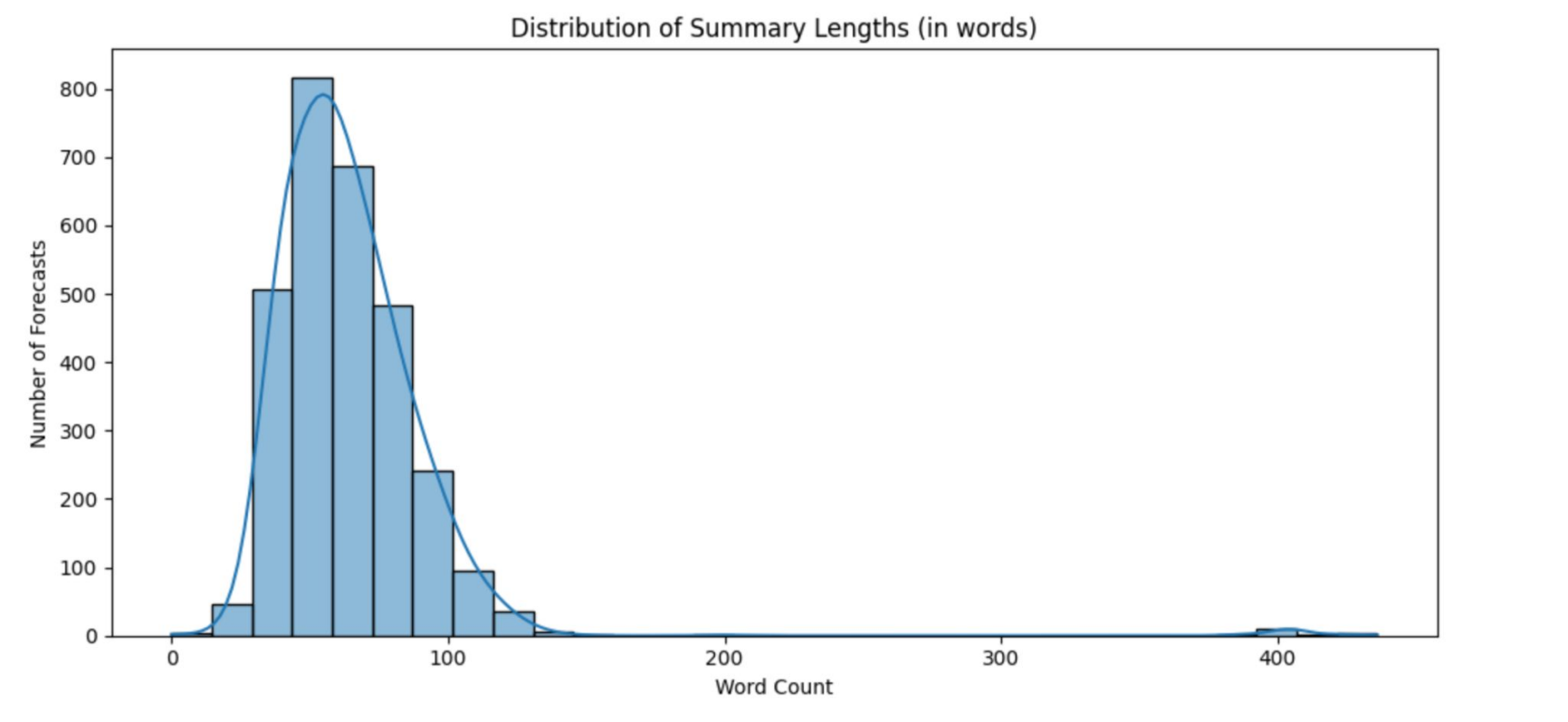
EDA - Top Words per Output (Above Treeline Only)



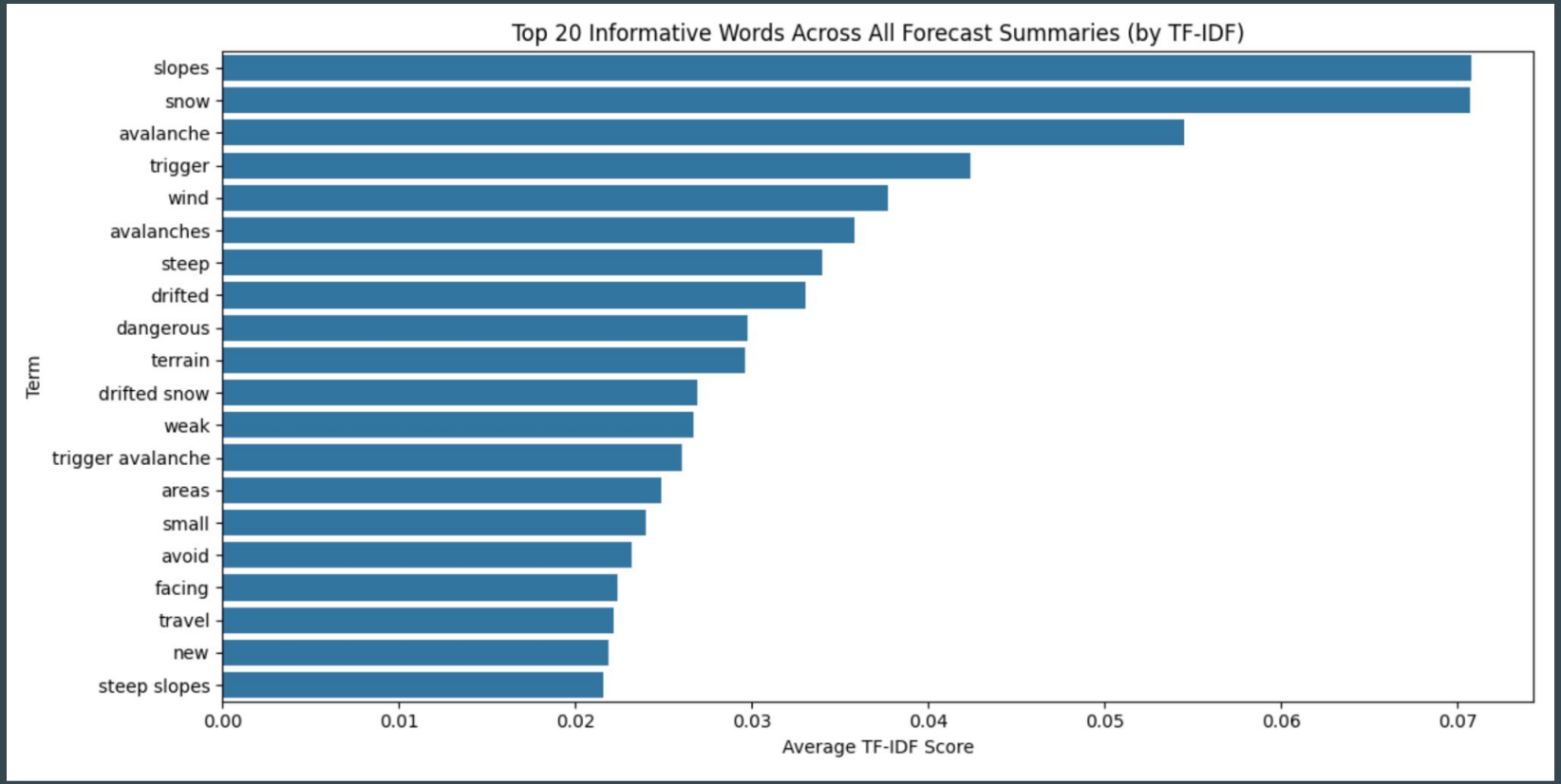




EDA - Summary Lengths



EDA - TF-IDF


Top 20 Informative Words Across All Forecast Summaries (by TF-IDF)
slopes
snow
avalanche
trigger
wind
avalanches
steep
drifted
dangerous
terrain
drifted snow
weak
trigger avalanche
areas
small
avoid
facing
travel
new
steep slopes
Term
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
Average TF-IDF Score

Baseline to Beat

I used sklearn's DummyClassifier with stratification to build a baseline for random predictions to aim to beat on accuracy scores:

```
Stratified DummyClassifier Accuracies:
  Below:     0.361
  Treeline:  0.313
  Above:     0.318
  Macro Avg: 0.330
```

Sequence vs Bag of Words

Based on rule of thumb from the book - ratio of number of samples and the mean number of words per sample:

"If that ratio is small—less than 1,500—then the bag-of-bigrams model will perform better (and as a bonus, it will be much faster to train and to iterate on too). If that ratio is higher than 1,500, then you should go with a sequence model."

That ratio for my data was very small at around 36, but I decided to continue with all different kinds of models anyway as a learning experience.

Text Preprocessing

One of the most confusing parts of my project was the text preprocessing involved. I needed to leverage different vectorizer parameters from tensorflow to make sure my text was in the correct format for the neural networks and consisted of the correct shapes. Similar to the textbook, my first models with dense layers used multi-hot encoding, and eventually with bigram and TF-IDF. The more complex models with embeddings used 'int' as the output mode to use the embedding models or trained embeddings. Done on the clean text data with all lowercase and stopwords and punctuation removed.

from tensorflow.keras.layers import TextVectorization

```
# preprocess text
vectorizer = TextVectorization(
    max_tokens=10000,
    output_mode='multi_hot',
    ngrams=None
)
```

Inputs

Model	Output Mode	Embedding?	Input Shape	Description
Bag of Words	‘multi_hot’	No	Vocabulary size: 2884	Ignores word order. Vector indicating which token is present.
Bag of Words - Extra Layer	‘multi_hot’	No	Vocabulary size: 2884	Ignores word order. Vector indicating which token is present.
Bigram	‘multi_hot’, ngrams=2	No	Vocabulary size: 10000	Ignores word order. Vector indicating which words and bigrams are present.
Bigram with TF-IDF	‘tf_idf’, ngrams=2	No	Vocabulary size: 10000	Ignores word order. Vector indicating with weights based on tf-idf per unigram/bigram.
Embedding Models	‘int’,output_sequence_length=126	Yes	output_sequence_length=126	Vector of integers representing tokens, passed to embedding layer.
Transformer Models	‘int’,output_sequence_length=126	Yes	output_sequence_length=126	Vector of integers representing tokens, passed to embedding layer.
DistliBERT	DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')	Yes	output_sequence_length=126	Vector of integers representing tokens, passed to embedding layer.

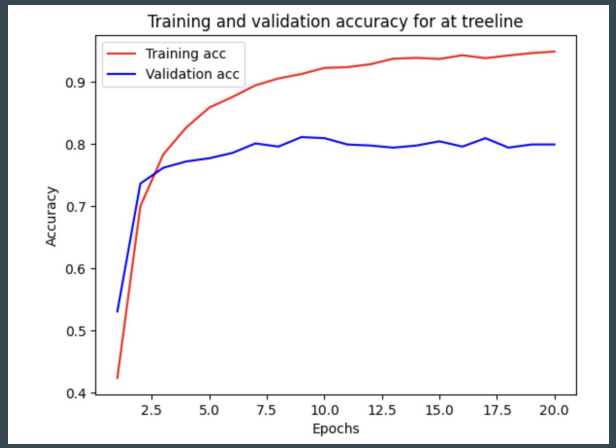
Outputs

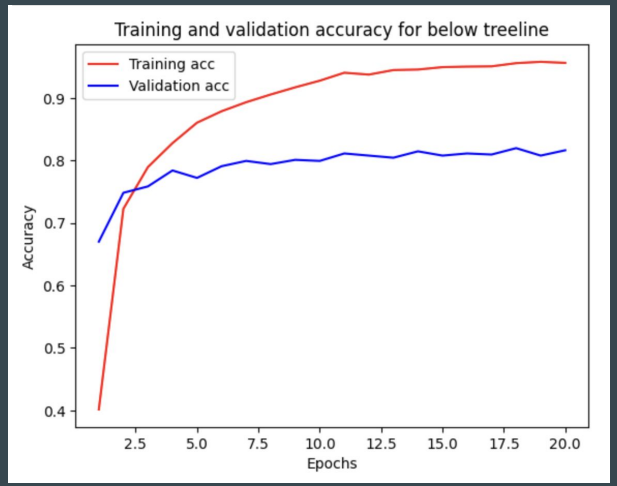
All of my models used the keras functional API with three outputs, one for each elevation tier and softmax activation functions for multi-class outputs.

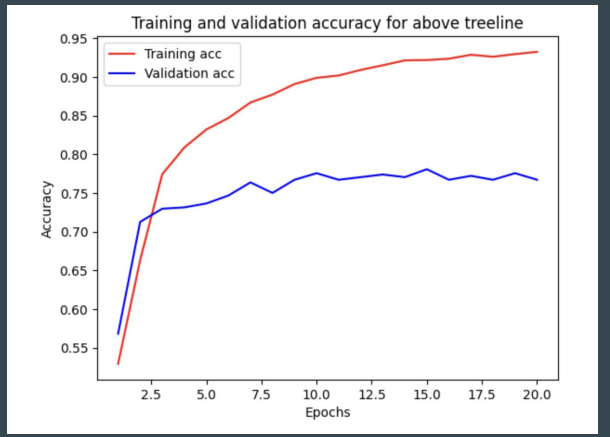
```
inputs = keras.Input(shape=(input_dim,))
x = layers.Dense(16, activation="relu")(inputs)
x = layers.Dense(16, activation='relu')(x)
x = layers.Dense(16, activation='relu')(x)

output_below = layers.Dense(num_classes, activation='softmax', name='below')(x)
output_treeline = layers.Dense(num_classes, activation='softmax', name='treeline')(x)
output_above = layers.Dense(num_classes, activation='softmax', name='above')(x)

model = keras.Model(inputs=inputs, outputs=[output_below, output_treeline, output_above])
```

Bag of Words - Basic Dense Layers

- Layers:
  - 2 Dense layers of 16 neurons each
- Outcome:
  - **Best val_above_accuracy: 0.7891**
  - **Best val_below_accuracy: 0.8146**
  - **Best val_treeline_accuracy: 0.8061**

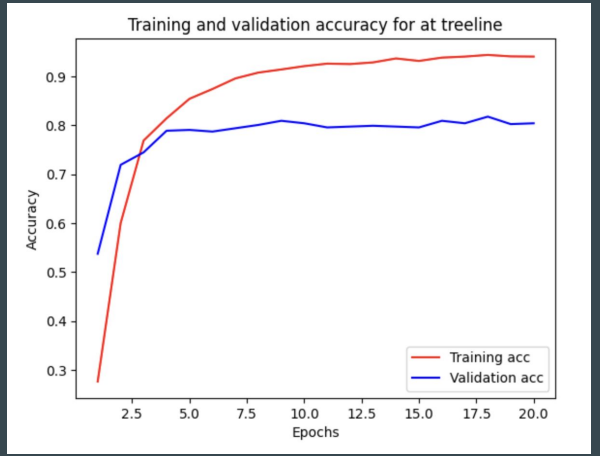


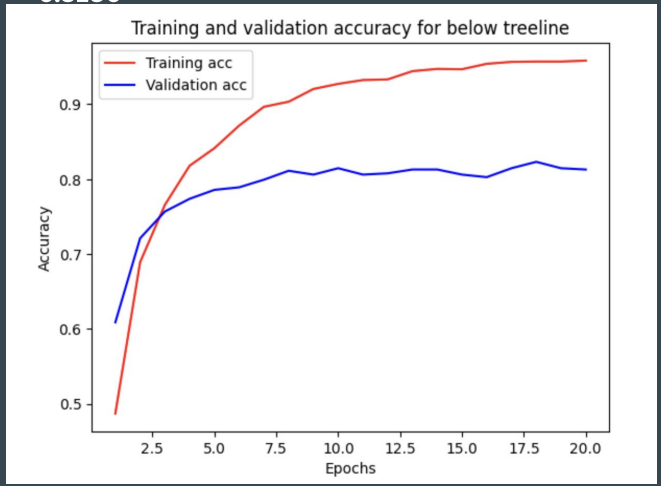


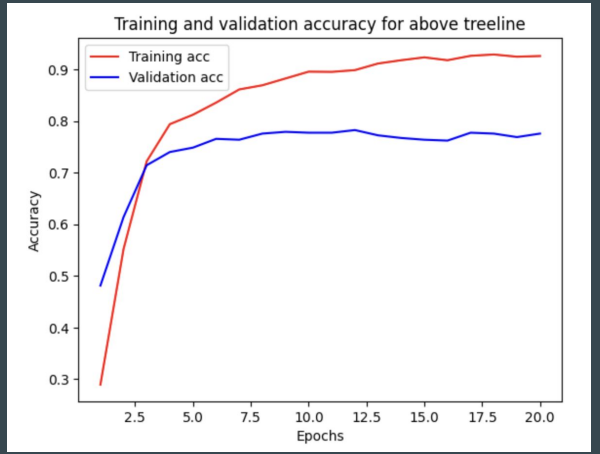


Bag of Words - Extra Layer

- Layers:
  - 3 Dense layers of 16 neurons each
- Outcome:
  - **Best val_above_accuracy: 0.7823**
  - **Best val_below_accuracy: 0.8231**
  - **Best val_treeline_accuracy: 0.8180**



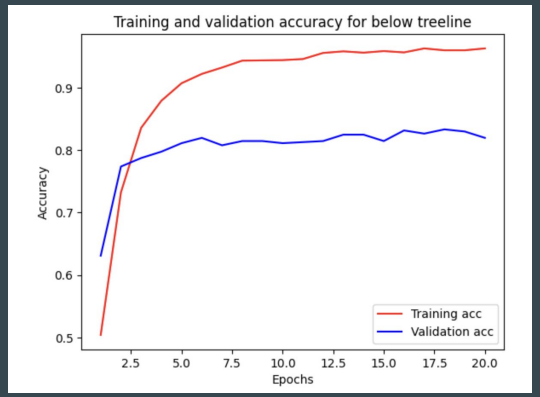


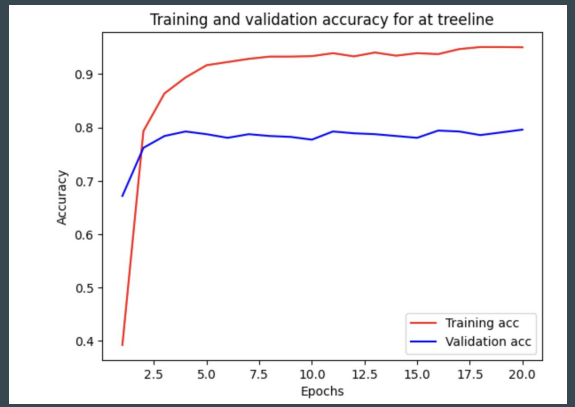


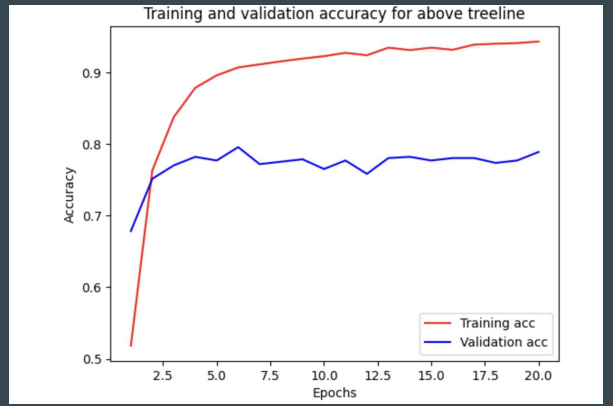
Bigram

- Layers:
  - 2 Dense layers of 16 neurons each
- Outcome:
  - **Best val_above_accuracy: 0.7959**
  - **Best val_below_accuracy: 0.8333**
  - **Best val_treeline_accuracy: 0.7959**

```
vectorizer = TextVectorization(
    max_tokens=10000,
    output_mode='multi_hot',
    ngrams=2
)
```



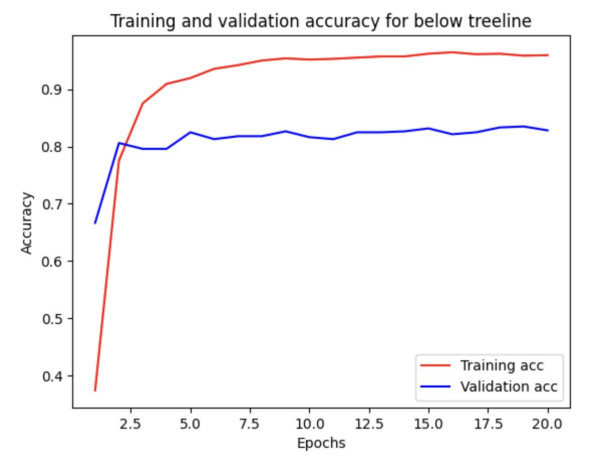


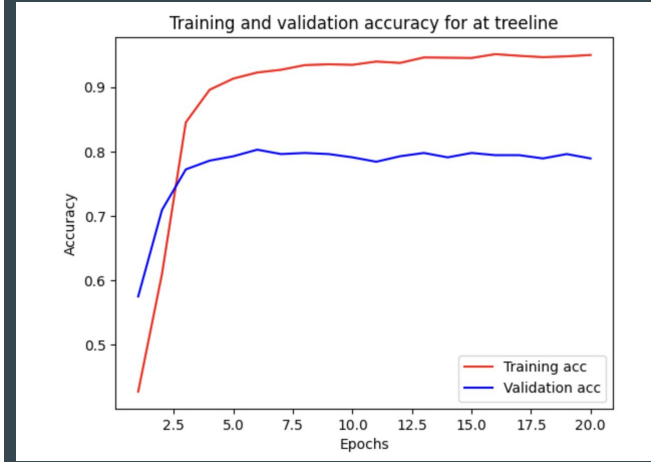


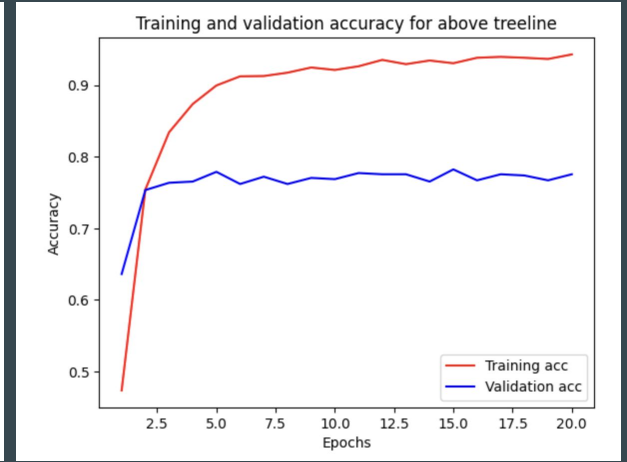
Bigram with TF-IDF

- Layers:
  - 2 Dense layers of 16 neurons each
- Outcome:
  - **Best val_above_accuracy: 0.7823**
  - **Best val_below_accuracy: 0.8350**
  - **Best val_treeline_accuracy: 0.8027**

```
4]: vectorizer = TextVectorization(
        max_tokens=10000,
        output_mode="tf_idf",
        ngrams=2
    )
```







Embedding Models

For the embedding models I based the sequence length on the following:

```
# check sequence lengths
df['text_length'] = df['summary_no_stop'].str.split().apply(len)
df['text_length'].describe()
```

```
count    2939.000000
mean       64.834297
std        32.795623
min         1.000000
25%        47.000000
50%        60.000000
75%        76.000000
max       436.000000
Name: text_length, dtype: float64
```

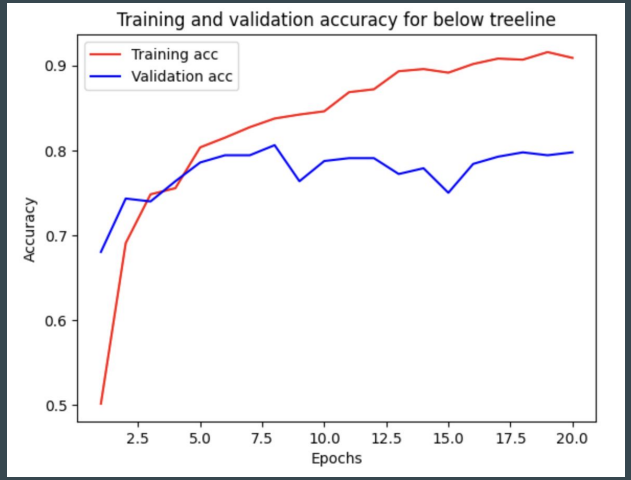
```
sequence_length = int(df['text_length'].quantile(.99))
sequence_length
```

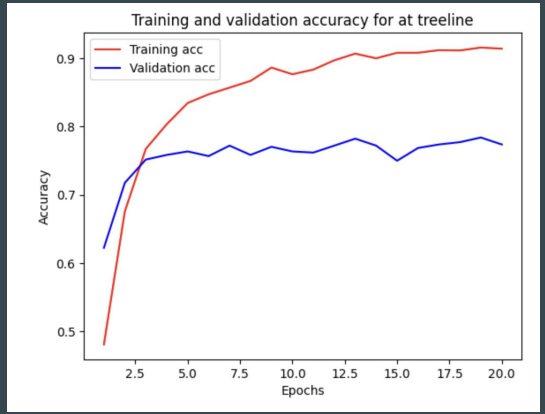
```
126
```

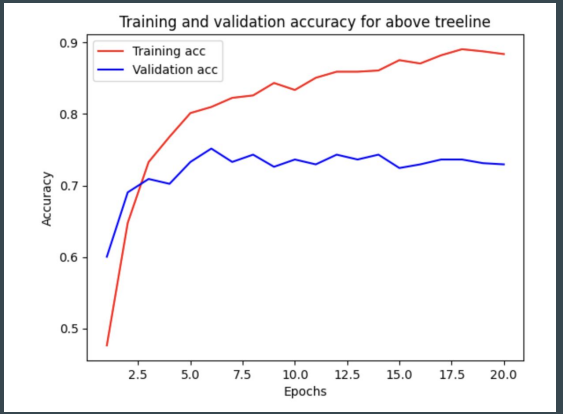
```
]: vectorizer = TextVectorization(
    max_tokens=10000,
    output_mode="int",
    output_sequence_length=126
)
```

Embedding with Bidirectional LSTM

- Layers:
  - Embedding Layer
  - Bidirectional LSTM with 32 neurons
  - 50% Dropout layer
- Outcome:
  - **Best val_above_accuracy: 0.7517**
  - **Best val_below_accuracy: 0.8061**
  - **Best val_treeline_accuracy: 0.7840**

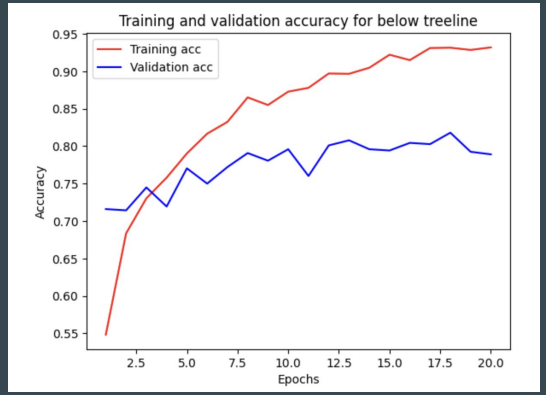


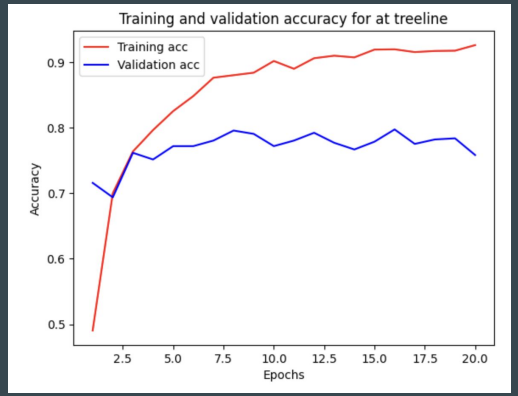


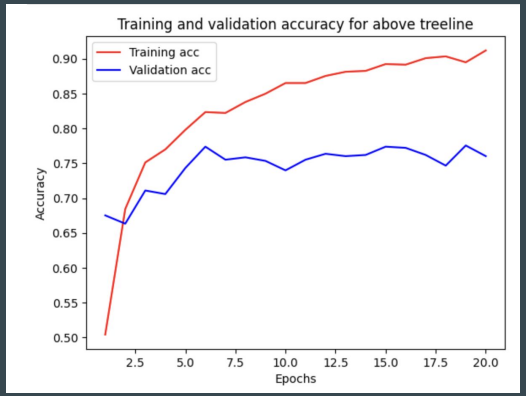


Embedding with Bidirectional LSTM - with Masking

- Layers:
  - Embedding Layer
  - Bidirectional LSTM with 32 neurons
  - 50% Dropout layer
  - mask_zero = True
- Outcome:
  - **Best val_above_accuracy: 0.7755**
  - **Best val_below_accuracy: 0.8180**
  - **Best val_treeline_accuracy: 0.7976**

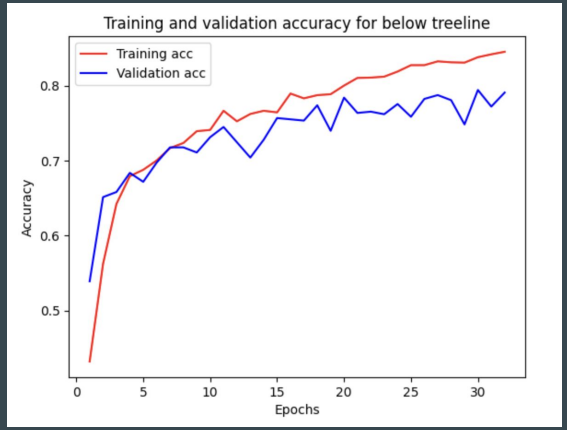


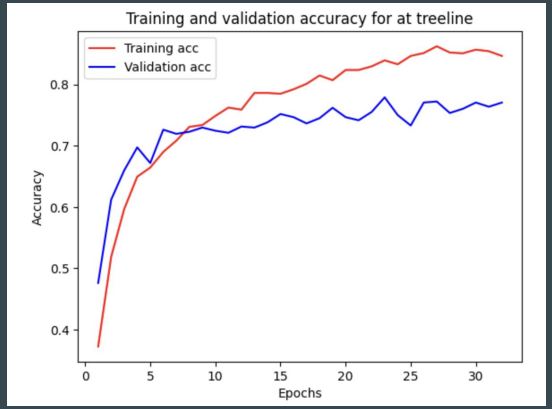


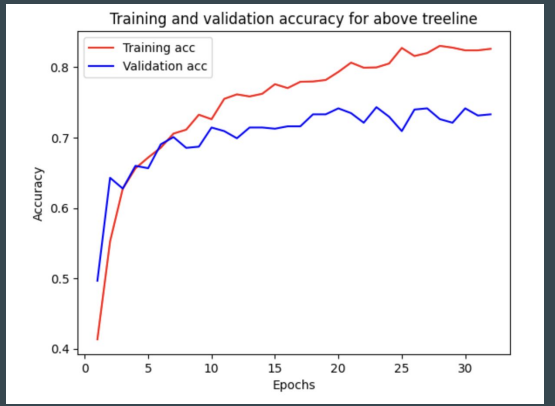


GloVe Word Embeddings

- Layers:
  - Embedding Layer - with Embedding Matrix
  - Bidirectional LSTM with 32 neurons
  - 50% Dropout layer
- Outcome:
  - **Best val_above_accuracy: 0.7432**
  - **Best val_below_accuracy: 0.7942**
  - **Best val_treeline_accuracy: 0.7789**

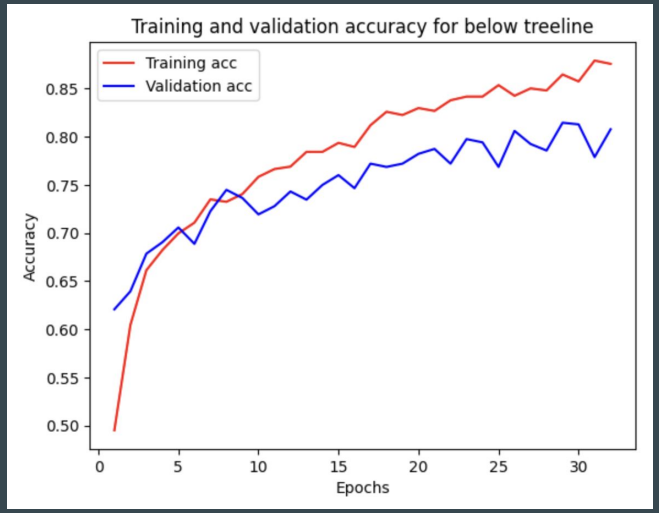


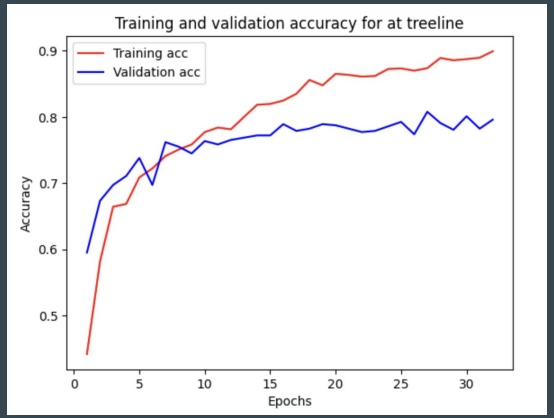


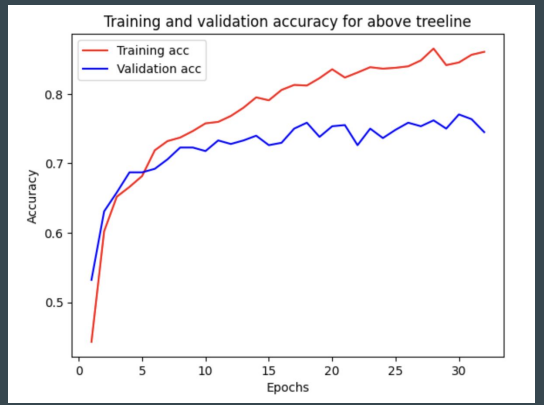


Word2Vec

- Layers:
  - Embedding Layer - with Embedding Matrix
  - Bidirectional LSTM with 32 neurons
  - 50% Dropout layer
- Outcome:
  - **Best val_above_accuracy: 0.7704**
  - **Best val_below_accuracy: 0.8146**
  - **Best val_treeline_accuracy: 0.8078**

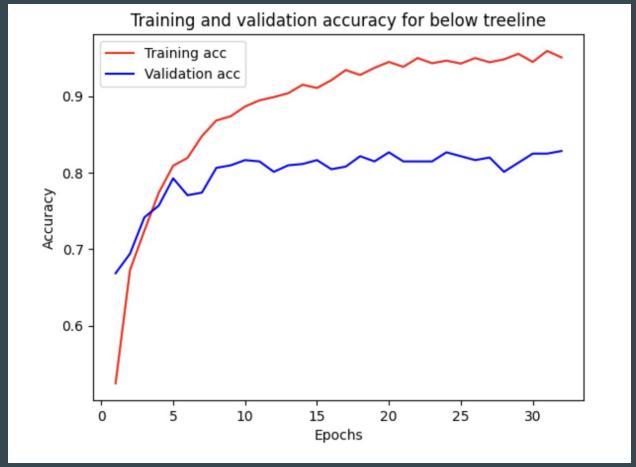


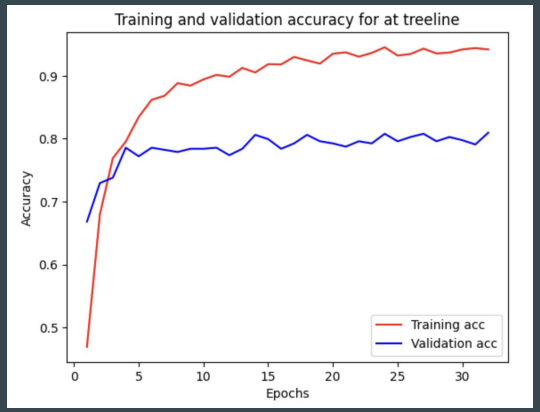


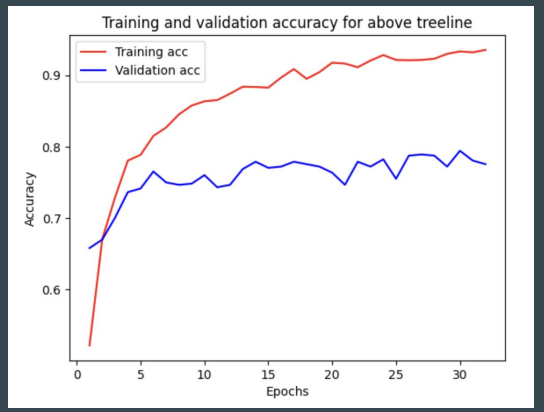


Word2Vec with Training

- Layers:
  - Embedding Layer - with Embedding Matrix
  - Bidirectional LSTM with 32 neurons
  - 50% Dropout layer
  - Embedding layer trainable = True
  - Embedding layer mask_zero = True
- Outcome:
  - **Best val_above_accuracy: 0.7942**
  - **Best val_below_accuracy: 0.8282**
  - **Best val_treeline_accuracy: 0.8095**

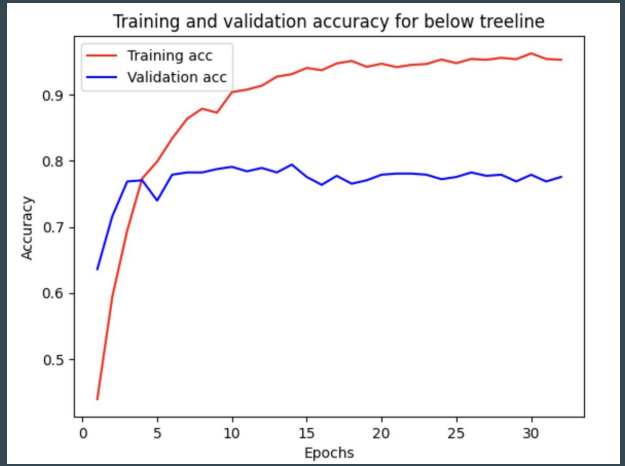


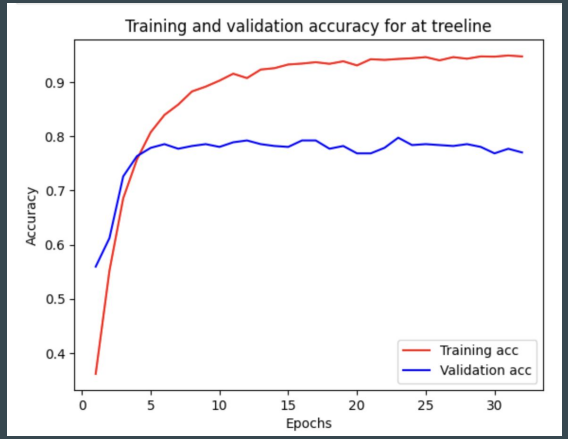


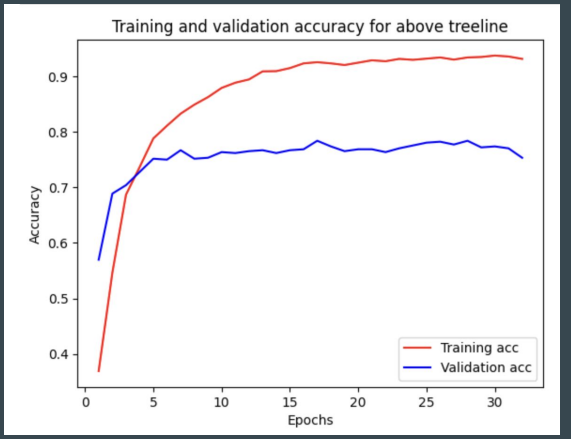


Transformer Encoder

- Layers:
  - Embedding Layer
  - Transformer Encoder layer (from Textbook):
    - MultiHeadAttention
    - Normalization layers
    - Dense layers
  - GlobalMaxPooling1D
  - 50% Dropout layer
- Outcome:
  - **Best val_above_accuracy: 0.7840**
  - **Best val_below_accuracy: 0.7942**
  - **Best val_treeline_accuracy: 0.7976**

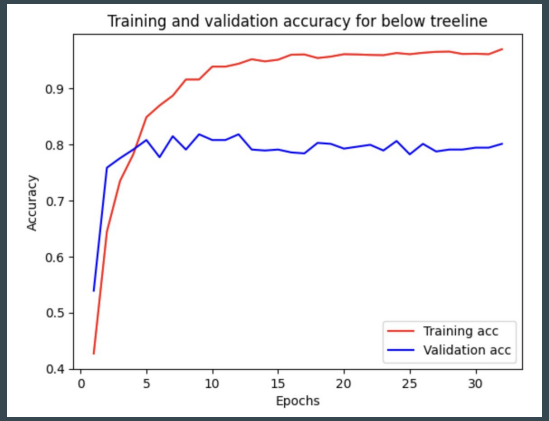


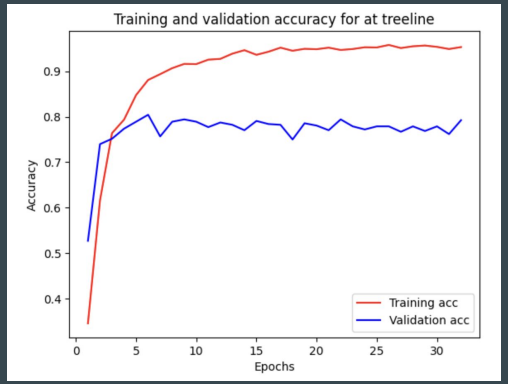


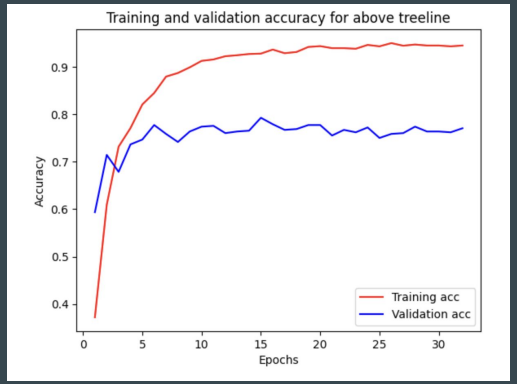


Transformer Encoder with Positional Embedding

- Layers:
  - PositionalEmbedding Layer (from Textbook)
  - Transformer Encoder layer (from Textbook):
    - MultiHeadAttention
    - Normalization layers
    - Dense layers
  - GlobalMaxPooling1D
  - 50% Dropout layer
- Outcome:
  - **Best val_above_accuracy: 0.7925**
  - **Best val_below_accuracy: 0.8180**
  - **Best val_treeline_accuracy: 0.8044**

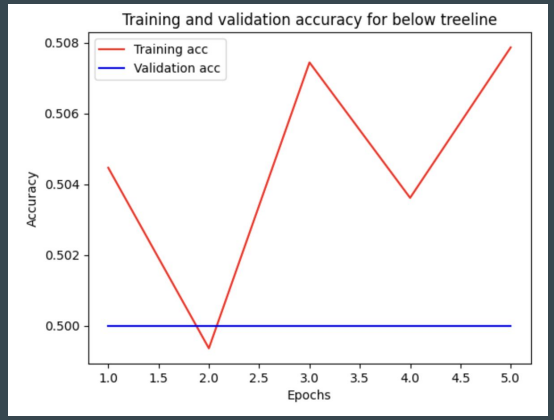


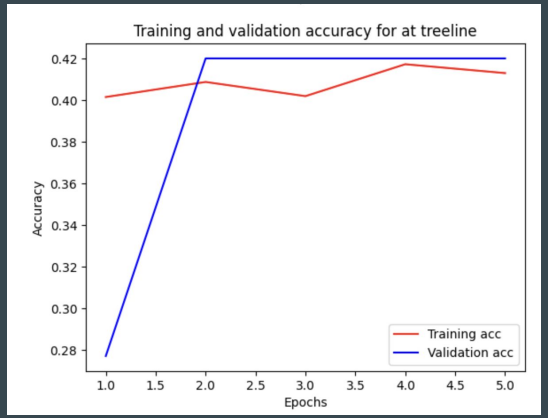


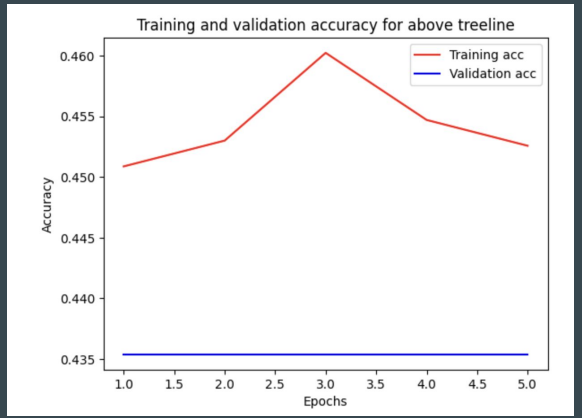


Pretrained Transformer - DistilBERT

- Layers:
  - DistilBert model
  - 50% dropout
- Outcome:
  - **Best val_tf.nn.softmax_accuracy: 0.5000**
  - **Best val_tf.nn.softmax_1_accuracy: 0.4201**
  - **Best val_tf.nn.softmax_2_accuracy: 0.4354**

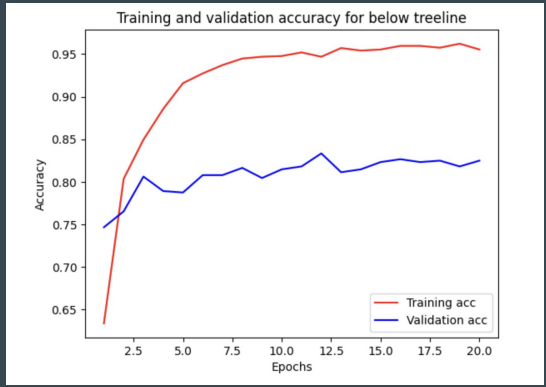


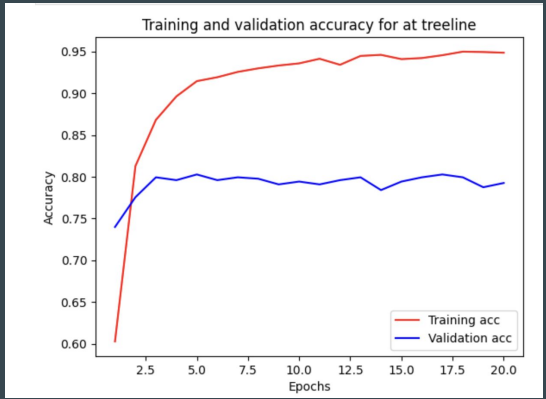


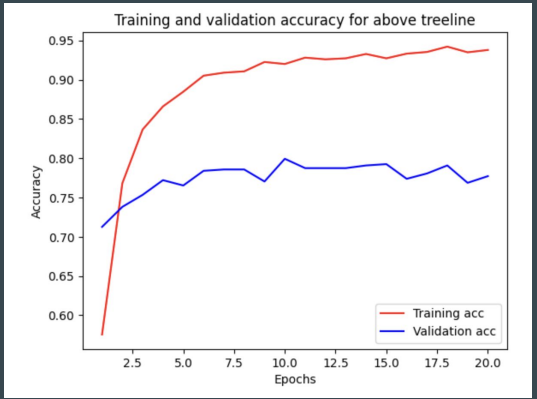


Bigram with TF-IDF - More Layers

- Layers:
  - 2 Dense layers of 16 neurons each
  - Dense layer of 32 neurons
  - Dense layer of 64 neurons
  - ‘rmsprop’ optimizer
- Outcome:
  - **Best val_above_accuracy: 0.7993**
  - **Best val_below_accuracy: 0.8333**
  - **Best val_treeline_accuracy: 0.8027**







Test Prediction

I used the last model to predict a more recent forecast:

text = "Start and end your day early to avoid wet snow avalanches. If overnight temperatures do not drop below freezing at 12,000 feet, steer clear of the steepest and rockiest terrain facing northwest through east (click here for the CAIC Weather Station page). In the last week, 23 avalanches were reported from across the state, and 25% of these were more dangerous and less predictable Wet Slab avalanches. They all occurred in very steep and rocky terrain, usually with bare ground near the start zone. Avoid this type of terrain late in the day or if the snowpack is unsupportable."

Actual:

Below treeline danger: 1
Treeline treeline danger: 1
Above treeline danger: 2

Predicted:

Below treeline danger: 2
Treeline treeline danger: 1
Above treeline danger: 1

Conclusion

Overall, these models performed much better than the baseline that I was aiming to beat, however there was not much difference in the outcome from all these different techniques, using bag of words vs sequence models or adding complexity with positional embedding and multi-head transformers. I think the main cause for this is the fact that my dataset was not very large. I only had around 3000 instances to train on from the CAIC vs 50000 in the IMBD example from the textbook. If I was able to continue this work I would request much more data, including examples with the **Extreme** danger level and across many years.

Thank You!