Appendix A
Extraction Methods

Notes on extraction methods of Google Ngram data:

NAA sequences:

For the Google Ngram data, downloading took place in two stages for both sets of sequences. For NAA, we initially downloaded 10 sequences from each file in the database (see adjectives_naa.py). This gave us 2011 sequences total, of which 879 were valid. Since we got 2011 sequences from a threshold of 10, to therefore increase the count of valid sequences to 1000, we needed at least 277 more sequences total to get 121 valid sequences. So we needed roughly two more sequences from each file, which we then downloaded separately (see adjectives_naa_2.py). This, together with the previously downloaded sequences gave us the total amount of 2599 sequences, all of which were classified, and of which 1008 were valid.

ANA sequences:

For ANA, we also initially downloaded 10 sequences from each file in the database. This gave us 2306 sequences total, of which 568 were valid. Since a much smaller percentage of these sequences were valid compared to the NAA sequences, before downloading more examples, we looked at the data to figure out why there was so much noise (invalid sequences) in the data. It turns out that a lot of the invalid sequences were due to English and French sequences classified by Google Ngram as Spanish ones, as well as "accentuated" prepositions or disjunctions (like 'á, ó, ú') classified as adjectives, in addition to proper names and 5 grams containing larger subsequences (like anaa or aana) classified as ANA sequences. So we decided to filter these out while downloading them (see adjectives_ana_2.py). Since we got 2306 sequences from a threshold of 10, to therefore increase the count of valid sequences to 1000, we needed at least 1754 more sequences total to get 432 valid sequences. Combined with the filters, we needed roughly 10 more sequences from each file, which we then downloaded separately (see adjectives_ana_2.py). This, together with the previously downloaded sequences gave us the total amount of 3813 sequences, of which 3499 were classified (due to time constraints), and of which 1214 were valid.

adjectives_naa.py

```
# From the google_ngram_downloader library, we import the readline_google_store function, which we will use to help download the sequences. The library was created by Dmitrijs Milajevs under the MIT License, and documentation can be found at https://pypi.org/project/google-ngram-downloader/

from google_ngram_downloader import readline_google_store

# The function that takes in the name of a file with ngrams ('aa', 'ab',…) and downloads naa sequences and stores them in output_file

def download_naa_3grams(indices,output_file):

    try:
        # Download 5 grams from the 2012 Google Ngram Corpus from a file.
        fname, url, records = next(readline_google_store(ngram_len=5, lang='spa',indices=indices))
        seq = set()

        # The number of sequences to be downloaded from each file.
        MAX_COUNT = 10
        count = 0

       # Open the file
        with open(output_file, 'w') as f:
            try:
                while count < MAX_COUNT:

                    # Get an ngram
                    record = next(records)
                    g1,g2,g3,g4,g5 = record.ngram.split()

                    # Check for naa sequences. Don't print naa sequences we have already seen.
                    if g1.endswith('_NOUN') and g2.endswith('_ADJ') and g3.endswith('_ADJ'):
                        if g1[:-5].isalpha() and g2[:-4].isalpha() and g3[:-4].isalpha():
                            if (g1,g2,g3) not in seq:
                                output =
[g1.split('_')[0],g2.split('_')[0],g3.split('_')[0],g4.split('_')[0],g5.split('_')[0]]
                                seq.add((g1,g2,g3))
                                print(output)
                                f.write(','.join(output).encode('utf-8') + '\n')
                                count += 1
                    elif g2.endswith('_NOUN') and g3.endswith('_ADJ') and g4.endswith('_ADJ'):
                        if g2[:-5].isalpha() and g3[:-4].isalpha() and g4[:-4].isalpha():
                            if (g2,g3,g4) not in seq:
                                output =
[g1.split('_')[0],g2.split('_')[0],g3.split('_')[0],g4.split('_')[0],g5.split('_')[0]]
                                seq.add((g2,g3,g4))
                                print(output)
                                f.write(','.join(output).encode('utf-8') + '\n')
                                count += 1
                    elif g3.endswith('_NOUN') and g4.endswith('_ADJ') and g5.endswith('_ADJ'):
                        if g3[:-5].isalpha() and g4[:-4].isalpha() and g5[:-4].isalpha():
                            if (g3,g4,g5) not in seq:
                                output =
[g1.split('_')[0],g2.split('_')[0],g3.split('_')[0],g4.split('_')[0],g5.split('_')[0]]
                                seq.add((g3,g4,g5))
                                print(output)
                                f.write(','.join(output).encode('utf-8') + '\n')
                                count += 1
            except StopIteration:
                pass
    except AssertionError:
        pass

def main():
    # For each file 'aa', 'ab', … ; download and print naa sequences.
    alphabet = []
    for letter in range(97,123):
        alphabet.append(chr(letter))
    for letter1 in alphabet:
        for letter2 in alphabet:
            index = letter1+letter2
            download_naa_3grams([index],index)
if __name__ == '__main__':
  main()
```

adjectives_naa_2.py

```
# From the google_ngram_downloader library, we import the readline_google_store function, which we will use to help
download the sequences. The library was created by Dmitrijs Milajevs under the MIT License, and documentation can be
found at https://pypi.org/project/google-ngram-downloader/

from google_ngram_downloader import readline_google_store

# The function that takes in the name of a file with ngrams ('aa', 'ab',…) and downloads naa sequences and stores them
in output_file

def download_naa_3grams(indices,output_file):

    try:
        # Download 5 grams from the 2012 Google Ngram Corpus from a file.
        fname, url, records = next(readline_google_store(ngram_len=5, lang='spa',indices=indices))
        seq = set()

        # The number of sequences to be downloaded from each file. Since we wanted to download an additional 2 examples
from each file, we increased the MAX_COUNT to 20 and then simply skipped the first 10 examples of each file as we had
already classified them.
        MAX_COUNT = 12
        count = 0

       # Open the file
        with open(output_file, 'w') as f:
            try:
                while count < MAX_COUNT:

                    # Get an ngram
                    record = next(records)
                    g1,g2,g3,g4,g5 = record.ngram.split()

                    # Check for naa sequences, skip the first ten ones. Don't print naa sequences we have already seen.
                    if g1.endswith('_NOUN') and g2.endswith('_ADJ') and g3.endswith('_ADJ'):
                        if g1[:-5].isalpha() and g2[:-4].isalpha() and g3[:-4].isalpha():
                            if (g1,g2,g3) not in seq:
                                output =
[g1.split('_')[0],g2.split('_')[0],g3.split('_')[0],g4.split('_')[0],g5.split('_')[0]]
                                seq.add((g1,g2,g3))
                                if count >= 10:
                                    print(output)
                                    f.write(','.join(output).encode('utf-8') + '\n')
                                count += 1
                    elif g2.endswith('_NOUN') and g3.endswith('_ADJ') and g4.endswith('_ADJ'):
                        if g2[:-5].isalpha() and g3[:-4].isalpha() and g4[:-4].isalpha():
                            if (g2,g3,g4) not in seq:
                                output =
[g1.split('_')[0],g2.split('_')[0],g3.split('_')[0],g4.split('_')[0],g5.split('_')[0]]
                                seq.add((g2,g3,g4))
                                if count >= 10:
                                    print(output)
                                    f.write(','.join(output).encode('utf-8') + '\n')
                                count += 1
                    elif g3.endswith('_NOUN') and g4.endswith('_ADJ') and g5.endswith('_ADJ'):
                        if g3[:-5].isalpha() and g4[:-4].isalpha() and g5[:-4].isalpha():
                            if (g3,g4,g5) not in seq:
                                output =
[g1.split('_')[0],g2.split('_')[0],g3.split('_')[0],g4.split('_')[0],g5.split('_')[0]]
                                seq.add((g3,g4,g5))
                                if count >= 10:
                                    print(output)
                                    f.write(','.join(output).encode('utf-8') + '\n')
                                count += 1
            except StopIteration:
                pass
    except AssertionError:
        pass

def main():
    # For each file 'aa', 'ab', … ; download and print naa sequences.
    alphabet = []
    for letter in range(97,123):
        alphabet.append(chr(letter))
```

```
    for letter1 in alphabet:
        for letter2 in alphabet:
            index = letter1+letter2
            download_naa_3grams([index],index)
if __name__ == '__main__':
  main()
```

adjectives_ana.py

```
# From the google_ngram_downloader library, we import the readline_google_store function, which we will use to help
download the sequences. The library was created by Dmitrijs Milajevs under the MIT License, and documentation can be
found at https://pypi.org/project/google-ngram-downloader/

from google_ngram_downloader import readline_google_store

# The function that takes in the name of a file with ngrams (‘aa’, ‘ab’,…) and downloads ana sequences and stores them
in output_file
def download_ana_3grams(indices,output_file):

    try:
        # Download 5 grams from the 2012 Google Ngram Corpus from a file.
        fname, url, records = next(readline_google_store(ngram_len=5, lang='spa',indices=indices))
        seq = set()

        # The number of sequences to be downloaded from each file.
        MAX_COUNT = 10
        count = 0

        # Open the file
        with open(output_file, 'w') as f:
            try:
                while count < MAX_COUNT:
                    # Get an ngram
                    record = next(records)
                    g1,g2,g3,g4,g5 = record.ngram.split()


                    output = [g1.split('_')[0],g2.split('_')[0],g3.split('_')[0],g4.split('_')[0],g5.split('_')[0]]

                   # Check for ana sequences. Don’t print ana sequences we have already seen.
                    if g1.endswith('_ADJ') and g2.endswith('_NOUN') and g3.endswith('_ADJ'):
                        if g1[:-4].isalpha() and g2[:-5].isalpha() and g3[:-4].isalpha():
                            if (g1,g2,g3) not in seq:
                                seq.add((g1,g2,g3))
                                print(output)
                                f.write(','.join(output).encode('utf-8') + '\n')
                                count += 1
                    elif g2.endswith('_ADJ') and g3.endswith('_NOUN') and g4.endswith('_ADJ'):
                        if g2[:-4].isalpha() and g3[:-5].isalpha() and g4[:-4].isalpha():
                            if (g2,g3,g4) not in seq:
                                output =
[g1.split('_')[0],g2.split('_')[0],g3.split('_')[0],g4.split('_')[0],g5.split('_')[0]]
                                seq.add((g2,g3,g4))
                                print(output)
                                f.write(','.join(output).encode('utf-8') + '\n')
                                count += 1
                    elif g3.endswith('_ADJ') and g4.endswith('_NOUN') and g5.endswith('_ADJ'):
                        if g3[:-4].isalpha() and g4[:-5].isalpha() and g5[:-4].isalpha():
                            if (g3,g4,g5) not in seq:
                                output =
[g1.split('_')[0],g2.split('_')[0],g3.split('_')[0],g4.split('_')[0],g5.split('_')[0]]
                                seq.add((g3,g4,g5))
                                print(output)
                                f.write(','.join(output).encode('utf-8') + '\n')
                                count += 1
            except StopIteration:
                pass
    except AssertionError:
        pass

def main():
    # For each file ‘aa’, ‘ab’, … ; download and print ana sequences.
    alphabet = []
    for letter in range(97,123):
        alphabet.append(chr(letter))
    for letter1 in alphabet:
        for letter2 in alphabet:
            index = letter1+letter2
            download_naa_3grams([index],index)
if __name__ == '__main__':
  main()
```

adjectives_ana_2.py

```
# From the google_ngram_downloader library, we import the readline_google_store function, which we will use to help
download the sequences. The library was created by Dmitrijs Milajevs under the MIT License, and documentation can be
found at https://pypi.org/project/google-ngram-downloader/

from google_ngram_downloader import readline_google_store

# The function that takes in the name of a file with ngrams (‘aa’, ‘ab’,…) and downloads ana sequences and stores them
in output_file
def download_ana_3grams(indices,output_file):

    try:
        # Download 5 grams from the 2012 Google Ngram Corpus from a file.
        fname, url, records = next(readline_google_store(ngram_len=5, lang='spa',indices=indices))
        seq = set()

        # The number of sequences to be downloaded from each file. Since we wanted to download an additional 10
examples from each file, we increased the MAX_COUNT to 20 and then simply skipped the first 10 examples of each file as
we had already classified them.
        MAX_COUNT = 20
        count = 0

        # Due to the large number of English and French words, as well as prepositions classified as adjectives found
in previous ana sequences, we decided to filter out those sequences by using the English and French words most commonly
found in the data, as well as the most common misclassified prepositions.
        english_words =
['the','of','in','and','this','is','to','are','as','at','for','its','on','that','from','be','by','but','was','it','that
','with','not','can','get','an','his','my','this','on','our']
        french_words = ['et','des','au','ne']
        accent_letters = [u'é',u'ó',u'ú',u'á']

        # Open the file
        with open(output_file, 'w') as f:
            try:
                while count < MAX_COUNT:
                    # Get an ngram
                    record = next(records)
                    g1,g2,g3,g4,g5 = record.ngram.split()


                    output = [g1.split('_')[0],g2.split('_')[0],g3.split('_')[0],g4.split('_')[0],g5.split('_')[0]]
                    # filter out English sequences
                    ew = [w for w in output if w in english_words]
                    if ew:
                        continue
                    # filter out French sequences
                    fw = [w for w in output if w in french_words]
                    if fw:
                        continue
                    # filter out accentuated single letters (misclassified prepositions)
                    al = [w for w in output if w in accent_letters]
                    if al:
                        continue

                   # Check for ana sequences, skip the first ten ones. Don’t print ana sequences we have already seen.
                    if g1.endswith('_ADJ') and g2.endswith('_NOUN') and g3.endswith('_ADJ'):
                        if g1[:-4].isalpha() and g2[:-5].isalpha() and g3[:-4].isalpha():
                            if (g1,g2,g3) not in seq:

                                # filter out proper names
                                pn = [w for w in output[:3] if w[0].isupper()]
                                if pn:
                                    continue

                                # filter out larger subsequences
                                if g4.endswith('_ADJ'):
                                    continue

                                seq.add((g1,g2,g3))
                                if count >= 10:
                                    print(output)
                                    f.write(','.join(output).encode('utf-8') + '\n')
                                count += 1
```

```
                    elif g2.endswith('_ADJ') and g3.endswith('_NOUN') and g4.endswith('_ADJ'):
                        if g2[:-4].isalpha() and g3[:-5].isalpha() and g4[:-4].isalpha():
                            if (g2,g3,g4) not in seq:
                                output =
[g1.split('_')[0],g2.split('_')[0],g3.split('_')[0],g4.split('_')[0],g5.split('_')[0]]

                                # filter out proper names
                                pn = [w for w in output[1:4] if w[0].isupper()]
                                if pn:
                                    continue

                                # filter out larger subsequences
                                if g1.endswith('_ADJ') or g5.endswith('_ADJ'):
                                    continue

                                seq.add((g2,g3,g4))
                                if count >= 10:
                                    print(output)
                                    f.write(','.join(output).encode('utf-8') + '\n')
                                count += 1
                    elif g3.endswith('_ADJ') and g4.endswith('_NOUN') and g5.endswith('_ADJ'):
                        if g3[:-4].isalpha() and g4[:-5].isalpha() and g5[:-4].isalpha():
                            if (g3,g4,g5) not in seq:
                                output =
[g1.split('_')[0],g2.split('_')[0],g3.split('_')[0],g4.split('_')[0],g5.split('_')[0]]

                                # filter out proper names
                                pn = [w for w in output[2:5] if w[0].isupper()]
                                if pn:
                                    continue

                                # filter out larger subsequences
                                if g2.endswith('_ADJ'):
                                    continue


                                seq.add((g3,g4,g5))
                                if count >= 10:
                                    print(output)
                                    f.write(','.join(output).encode('utf-8') + '\n')
                                count += 1
            except StopIteration:
                pass
    except AssertionError:
        pass

def main():
    # For each file 'aa', 'ab', … ; download and print ana sequences.
    alphabet = []
    for letter in range(97,123):
        alphabet.append(chr(letter))
    for letter1 in alphabet:
        for letter2 in alphabet:
            index = letter1+letter2
            download_naa_3grams([index],index)
if __name__ == '__main__':
  main()
```