CS158_HW4___

November 30, 2020

[522]:

```
import numpy as np
from numpy.linalg import norm
import math
from urllib.request import urlopen
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from urllib.request import urlopen
import random

import urllib
import scipy.optimize
import random
from collections import defaultdict # Dictionaries with default values
import nltk
from nltk.util import ngrams
import string
from nltk.stem.porter import *
import ast

import gzip
from collections import defaultdict
```

[ ]:

```

```

[406]:

```
def parseDataFromURL(fname):
    for l in urlopen(fname):
        yield eval(l)

def parseData(fname):
    for l in open(fname):
        yield eval(l)
```

[ ]:

```
print("Reading data...")
data = list(parseData("./assignment1/train_Category.json"))
print("done")
```

```

df = pd.DataFrame(data)
df
```

[408]:

```
training_data = data[0:10000]
training_data[0]
```

[408]:

```
{'userID': 'u74382925',
 'genre': 'Adventure',
 'early_access': False,
 'reviewID': 'r75487422',
 'hours': 4.1,
 'text': 'Short Review:\nA good starting chapter for this series, despite the
main character being annoying (for now) and a short length. The story is good
and actually gets more interesting. Worth the try.\nLong Review:\nBlackwell
Legacy is the first on the series of (supposedly) 5 games that talks about the
main protagonist, Rosangela Blackwell, as being a so called Medium, and in this
first chapter we get to know how her story will start and how she will meet her
adventure companion Joey…and really, that\'s really all for for now and
that\'s not a bad thing, because in a way this game wants to show how hard her
new job is, and that she cannot escape her destiny as a medium.\nMy biggest
complain for this chapter, except the short length, it\'s the main protagonist
being a "bit" too annoying to be likeable, and most of her dialogues will always
be about complaining or just be annoyed. Understandable, sure, but lighten\' up
will ya!?\nHowever, considering that in the next installments she will be much
more likeable and kind of interesting, I\'d say give it a shot and see if you
like it: if you hate this first game, you might like the next, or can always
stop here.\nI recommend it.',
 'genreID': 3,
 'date': '2014-02-07'}
```

0.1 1.

[409]:

```
BigramCount = defaultdict(int)
totalBigrams = 0

punct = string.punctuation


for d in training_data:
    t = d['text']
    t = t.lower() # lowercase string
    t = [c for c in t if not (c in punct)] # non-punct characters
    t = ''.join(t) # convert back to string
    words = t.strip().split() # tokenizes
    for i in range(len(words)-1):
        totalBigrams += 1
        BigramCount[(words[i], words[i+1])] += 1
```

[410]:

```
counts = [(BigramCount[b], b) for b in BigramCount]
counts.sort()
counts.reverse()
print('Unique Bigrams:',len(counts))

print('\nTop 5 most frequent Bigrams')
counts[:5]
```

```
Unique Bigrams: 256618

Top 5 most frequent Bigrams
```

[410]:

```
[(4441, ('this', 'game')),
 (4249, ('the', 'game')),
 (3359, ('of', 'the')),
 (2020, ('if', 'you')),
 (2017, ('in', 'the'))]
```

[ ]:

```

```

0.2 2.

[411]:

```
bigrams = [b[1] for b in counts[:1000]]
bigramId = dict(zip(bigrams, range(len(bigrams))))
bigramSet = set(bigrams)
print(len(bigramSet))
```

```
1000
```

[412]:

```
def feature(datum):
    feat = [0]*len(bigramSet)
    t = datum['text']
    t = t.lower() # lowercase string
    t = [c for c in t if not (c in punct)] # non-punct characters
    t = ''.join(t) # convert back to string
    words = t.strip().split() # tokenizes
    for i in range(len(words)-1):
        b = (words[i], words[i+1])
        if not (b in bigramSet): continue
        feat[bigramId[b]] += 1
    feat.append(1)
    return feat
```

[413]:

```
X = [feature(d) for d in training_data]
y = [np.log2(d['hours'] + 1) for d in training_data]
```

[414]:
```
clf = linear_model.Ridge(1.0, fit_intercept=False) # MSE + 1.0 l2
clf.fit(X, y)
theta = clf.coef_
predictions = clf.predict(X)
```

[415]:
```
MSE = sum((predictions - y)**2) / len(y)
print('MSE =', MSE)
```

```
MSE = 4.39378724720003
```

[ ]:
```
```

0.3 3.

[416]:
```
# Get top 1000 unigrams
unigramCount = defaultdict(int)
totalUnigrams = 0

for d in training_data:
    t = d['text']
    t = t.lower() # lowercase string
    t = [c for c in t if not (c in punct)] # non-punct characters
    t = ''.join(t) # convert back to string
    words = t.strip().split() # tokenizes
    for w in words:
        totalUnigrams += 1
        unigramCount[w] += 1
```

[417]:
```
countsUni = [(unigramCount[w], w) for w in unigramCount]
countsUni.sort()
countsUni.reverse()
print(len(countsUni))
```

```
29692
```

[418]:
```
# Strategy: Combine top 500 unigrams and top 500 bigrams together
unigrams = [w[1] for w in countsUni[:500]]
unigramId = dict(zip(unigrams, range(len(unigrams))))
unigramSet = set(unigrams)
print(len(unigramSet))

bigrams = [b[1] for b in counts[:500]]
bigramId = dict(zip(bigrams, range(len(unigrams), len(unigrams) + len(bigrams))))
bigramSet = set(bigrams)
print(len(bigramSet))
```

```
500
500
```

[419]:
```
def feature2(datum):
    feat = [0]*(len(unigramSet) + len(bigramSet))
    t = datum['text']
    t = t.lower() # lowercase string
    t = [c for c in t if not (c in punct)] # non-punct characters
    t = ''.join(t) # convert back to string
    words = t.strip().split() # tokenizes
    for w in words:
        if not (w in unigramSet): continue
        feat[unigramId[w]] += 1

    for i in range(len(words)-1):
        b = (words[i], words[i+1])
        if not (b in bigramSet): continue
        feat[bigramId[b]] += 1
    feat.append(1)
    return feat
```

[420]:
```
X = [feature2(d) for d in training_data]
y = [np.log2(d['hours'] + 1) for d in training_data]
```

[421]:
```
clf = linear_model.Ridge(1.0, fit_intercept=False) # MSE + 1.0 l2
clf.fit(X, y)
theta = clf.coef_
predictions = clf.predict(X)
```

[422]:
```
MSE = sum((predictions - y)**2) / len(y)
print('MSE =', MSE)
```

```
MSE = 4.249894468206029
```

[ ]:
```

```

0.4 4.

[423]:
```
def idf(word, data):
    counter = 0
    for datum in data:
        t = datum['text']
        t = t.lower() # lowercase string
        t = [c for c in t if not (c in punct)] # non-punct characters
        t = ''.join(t) # convert back to string
        words = t.strip().split() # tokenizes
        if word in words:
            counter += 1

    return np.log10(len(data) / counter)
```

[424]:

```
idf_words = ['destiny', 'annoying', 'likeable', 'chapter', 'interesting']

for w in idf_words:
    idf_val = idf(w, training_data)
    print('IDF for ' + w + ' =', idf_val)
```

```
IDF for destiny = 3.3979400086720375
IDF for annoying = 1.8386319977650252
IDF for likeable = 3.0969100130080562
IDF for chapter = 2.221848749616356
IDF for interesting = 1.3585258894959005
```

[445]:

```
# tf-idf of the five words in a document
def tf_idf(word, datum, data):
    counter = 0
    t = datum['text']
    t = t.lower() # lowercase string
    t = [c for c in t if not (c in punct)] # non-punct characters
    t = ''.join(t) # convert back to string
    words = t.strip().split() # tokenizes

    for w in words:
        if w == word:
            counter += 1
    return counter * idf(word, data)
```

[426]:

```
for d in training_data:
    if d['reviewID'] == 'r75487422':
        for w in idf_words:
            tf_idf_val = tf_idf(w, d, training_data)
            print('Tf-Idf for \'' + w + '\' =', tf_idf_val)
        break
```

```
Tf-Idf for 'destiny' = 3.3979400086720375
Tf-Idf for 'annoying' = 3.6772639955300503
Tf-Idf for 'likeable' = 6.1938200260161125
Tf-Idf for 'chapter' = 6.665546248849068
Tf-Idf for 'interesting' = 2.717051778991801
```

[ ]:

0.5 5.

[511]:

```python
unigrams = [w[1] for w in countsUni[:1000]]
unigramId = dict(zip(unigrams, range(len(unigrams))))
unigramSet = set(unigrams)
print(len(unigramSet))

# Map to improve runtime of idf
def getIdfMap(unigramSet, data):
    idfCounter = defaultdict(float)
    for datum in data:
        t = datum['text']
        t = t.lower() # lowercase string
        t = [c for c in t if not (c in punct)] # non-punct characters
        t = ''.join(t) # convert back to string
        words = t.strip().split() # tokenizes
        for w in words:
            if w in unigramSet:
                idfCounter[w] += 1
    return idfCounter

idfCounter = getIdfMap(unigramSet, training_data)
```

```
1000
```

[512]:

```python
# tf-idf using faster implementation of idf
def tf_idf2(word, datum, data):
    counter = 0
    t = datum['text']
    t = t.lower() # lowercase string
    t = [c for c in t if not (c in punct)] # non-punct characters
    t = ''.join(t) # convert back to string
    words = t.strip().split() # tokenizes

    for w in words:
        if w == word:
            counter += 1
    return counter * np.log10(len(data) / idfCounter[word])
```

[513]:

```python
# Unigrams only
def feature3(datum):
    feat = [0.0]*(len(unigramSet))
    t = datum['text']
    t = t.lower() # lowercase string
    t = [c for c in t if not (c in punct)] # non-punct characters
    t = ''.join(t) # convert back to string
    words = t.strip().split() # tokenizes
    for w in words:
        if not (w in unigramSet): continue
        feat[unigramId[w]] = tf_idf2(w, datum, training_data)
    feat.append(1)
    return feat
```

[514]:

```
X = [feature3(d) for d in training_data]
y = [np.log2(d['hours'] + 1) for d in training_data]
```

[515]:

```
clf = linear_model.Ridge(1.0, fit_intercept=False) # MSE + 1.0 l2
clf.fit(X, y)
theta = clf.coef_
predictions = clf.predict(X)
```

[516]:

```
MSE = sum((predictions - y)**2) / len(y)
print('MSE =', MSE)
```

```
MSE = 4.199776205842035
```

[ ]:

```

```

0.6 6.

[538]:

```
def CosineSimilarity(a, b):
    return np.dot(a,b) / (norm(a) * norm(b))
```

[539]:

```
r75487422_data = []

for datum in training_data:
    if datum['reviewID'] == 'r75487422':
        r75487422_data = datum
        break

r75487422_tf_idf = feature3(r75487422_data)


best_cos_sim = -99999
best_datum = []

for datum in training_data:
    if datum['reviewID'] == 'r75487422':
        continue

    tf_idf_rep = feature3(datum)
    cos_sim = CosineSimilarity(r75487422_tf_idf, tf_idf_rep)

    if best_cos_sim == -99999:
        best_cos_sim = cos_sim
```

```python
        best_datum = datum
    elif cos_sim > best_cos_sim:
        best_cos_sim = cos_sim
        best_datum = datum


print('ReviewID = r75487422\n')
print('Review Text:\n', r75487422_data['text'])

print('\n\n')
print('Best Cosine Similarity compared to r75487422\n')
print('ReviewID =', best_datum['reviewID'])
print('Review Text:\n', best_datum['text'])
print('\n')
print('Cosine Similarity =', best_cos_sim)
```

```
ReviewID = r75487422

Review Text:
 Short Review:
A good starting chapter for this series, despite the main character being
annoying (for now) and a short length. The story is good and actually gets more
interesting. Worth the try.
Long Review:
Blackwell Legacy is the first on the series of (supposedly) 5 games that talks
about the main protagonist, Rosangela Blackwell, as being a so called Medium,
and in this first chapter we get to know how her story will start and how she
will meet her adventure companion Joey…and really, that's really all for for
now and that's not a bad thing, because in a way this game wants to show how
hard her new job is, and that she cannot escape her destiny as a medium.
My biggest complain for this chapter, except the short length, it's the main
protagonist being a "bit" too annoying to be likeable, and most of her dialogues
will always be about complaining or just be annoyed. Understandable, sure, but
lighten' up will ya!?
However, considering that in the next installments she will be much more
likeable and kind of interesting, I'd say give it a shot and see if you like it:
if you hate this first game, you might like the next, or can always stop here.
I recommend it.



Best Cosine Similarity compared to r75487422

ReviewID = r33215456
Review Text:
 Before I tell you about RimWorld, let me tell you a story that happened in
RimWorld. It’s about that girl up there, drinking a beer. If you’re not
```

convinced to jump in by the end of this tale, then we have nothing more to talk about. We can't be friends. Everyone else: we're still cool. So here it is, the story of Min, a pop star with a privileged upbringing, who is about to come crashing down to earth.

Min slept with all her animals around her. Since the crash, the 16-year-old pop idol was finding it hard to adjust. She would not help with heavy lifting, or farming, or pulling up weeds. The only thing she would do is take care of the colony's animals. Rabbits, dogs, tortoises. She loved them all. She even formed a close bond with a rat, whom she named 'Illness'.

But she wasn't alone. In this game, you can create your own scenario before crash-landing onto the planet's surface, adding or removing starter items (wood, steel, pelts, gold, widescreen TVs). You can also set the number of colonists and tweak dozens of other small rules, like compulsory character traits or rules about the planet's climate. The game let's you mould your scenario completely, right down to the flavour text.

As you can see, my five colonists had all escaped from a crashing party yacht. They were all rich, famous or both. A jeweler, a wealthy businessman, an ace fighter pilot, a star surgeon and (of course) Min the pop star. They all landed with severe hangovers. I am not kidding, this is a setting you can change.

Fast forward some weeks. The colonists had settled in. They had even survived their first few pirate raids. During one of these raids, they captured a large pirate with a low IQ called Ferdnand. This man was physically capable of nothing but sculpting statues and communing with animals. He was like Hodor with a paintbrush. FERDNAND! Eventually, we convinced him to join the colony.

I set him the task of taming a Megatherium - a huge and powerful ground sloth - so that the colony would have some extra muscle. But Min the pop idol was not having it. She had always been the colony's animal trainer, and she had little 'Illness' to prove it. Spurned, she decided to take on the job herself. She left the safety of the village walls and approached the huge beast with some corn in hand. It went mad.

Ferdnand was already on his way to train his new would-be pet when he saw Min being savaged. He took out his Uzi and started firing with surprising accuracy. FERDNAND! Every shot landed. But the Megatherium turned and charged. By the time Ferdnand started to run, it was too late. The animal ripped him to pieces and he died on the spot.

The other colonists formed up. They lured the beast into the walls, where they riddled it with bullets until dead. The colony's nudist surgeon (who weeks ago had experienced a mental breakdown and now did everything naked to keep himself happy) rushed out to rescue Min, who was lying in a bloody heap, writhing in unbearable pain. He took her to the clinic, wrapped her in bandages and made sure she got everything she needed - medicine, food and rest.

Well, maybe not rest.

You see, I had already scheduled the clinic's floor to be renovated during this time. And as Min slept (or tried to sleep) my other colonists were busy ripping up planks and replacing them with shiny while tiles, sending sparks and wood chips everywhere. The noise was unbearable. Min snapped.

Crazed with pain and lack of sleep, she went berserk and chased everyone out of the clinic. She terrorised the dinner guests in the common room, and pursued them into the garden, where she chased them around. When she couldn't catch the colonists, she turned on the animals - hares, pigs and rats - punching the poor creatures to death in her rage, ignoring the scratches and bites she got in return. Ignoring her old friends.

Finally, a colonist stepped up, walking up to the rampaging diva and knocking her out cold. When Min came to, she was back in the clinic. It was completely refurbished, clean and presentable. But the same could not be said for Min. This once beautiful pop idol, who people would fall in love with before she even started to sing, was in tatters. She now had no nose, no left ear and no right arm.

Over the next two weeks she locked herself in her room and fell into terrible moods. She stopped handling the animals and was assigned to work only on sculptures. Her first work of art depicted a chicken vomiting on the shoes of a lawyer. I put it in the centre of her workshop.

One day, a ship appeared in orbit. They had items to trade. Among them was a prosthetic limb called a "power claw". After some thought, I decided it was better than nothing. And besides, these rich castaways could afford it. The nudist was scheduled to do the surgery. Min was getting her arm back, even if it was a claw.

When she woke this time, she went straight to her room and locked herself in. I wondered what was wrong. It wasn't until she was back at her post, carving new statues that I realised what had happened.

We had amputated the wrong arm.

Now, not only was Min a noseless, earless, armless monster, she also had a terrifying claw instead of her only remaining hand.

Welcome to RimWorld, where stories like this pointless tragedy are commonplace. It can probably best be summed up as: "the game for everyone who wanted to get into Dwarf Fortress but couldn't because ASCII." That's not say it doesn't have its own share of confusing menus. Supposedly there is a fluid teaching system but it seemed more to me like hints handed out at random. In the absence of a proper tutorial, it takes a lot of clicking and umming to understand why everything functions the way it does.

Let's look at the colonists, for instance. For the most part, they run around of their own volition but you can still take direct control in a number of ways. When I said that Ferdnand took out his gun, I meant that I clicked on Ferdnand and clicked the "draft" button, putting him into military mode - something you'll need to survive raids and attacks. But when I said he was already on his way to the gigantic animal, I meant he was heading to it autonomously, because it had been assigned to be "tamed" and he was a designated animal handler.

There's a task grid that gives some order to it all. You can select which jobs you want each individual colonist to do and which jobs you want them to ignore. An advanced version lets you organise these even further by priority, fiddling with the numbers until you have each person specialised in a strict set of tasks. This is hidden away under "manual" control but it is something that soon becomes essential to learn.

It looks far more complicated than it is. And at any time you can just click on a colonist and right-click on something you want "prioritised" to have them commit to this instead of their job. It's a bit fiddly but comes in useful in a

```
pinch, for example, when someone insists on milking a cow in a burning barn
instead of putting the fire out.
Building your settlement is a lot more straightforward. Prison Architect is the
obvious inspiration for the way much of RimWorld’s construction and furniture
placement works. You drag lines of walls made of wood, steel or stone until you
have what you want, filling it with nice floors, beds, billiards tables, lamps
and so on. Power is supplied by solar panels, windmills, wood-burning
generators. And electricity is stored in batteries and disseminated by carefully
laid power conduits. Sometimes these power cables like to explode, sending your
whole colony into darkness and spoiling all the food in your freezer. Sometimes
the batteries get wet and go on fire. Sometimes that fire spreads to your
sleeping quarters and burns everybody to death. How whimsical!
It’s these accidents and mistakes that make it much more than just another
management sim. And coupled with the finer details of the simulation, it makes
for som


Cosine Similarity = 0.49042613059213225
```

[ ]:

```

```

0.7 7.

[706]:

```python
# Shuffle data
random.shuffle(data)

training_data = data[:10000]
validation_data = data[10000:20000]
test_data = data[20000:30000]

punct = string.punctuation
```

[650]:

```python
# Gets IDF Counts of Unigram (Remove or Preserve Punctuation)
def getIdfMapUnigram(unigramSet, data, rp):
    idfCounter = defaultdict(float)
    for datum in data:
        t = datum['text']
        t = t.lower() # lowercase string
        if rp == True:
            t = [c for c in t if not (c in punct)] # non-punct characters

        # Keep punctuation
        else:
            new_arr = []
            for c in t:
                if not (c in punct):
                    new_arr.append(c)
                else:
                    new_arr.extend([' ', c, ' '])
            t = new_arr
        t = ''.join(t) # convert back to string
        words = t.strip().split() # tokenizes
        for w in words:
            if w in unigramSet:
                idfCounter[w] += 1
    return idfCounter

# Gets IDF Counts of Bigram (Remove or Preserve Punctuation)
def getIdfMapBigram(bigramSet, data, rp):
    idfCounter = defaultdict(float)
    for datum in data:
        t = datum['text']
        t = t.lower() # lowercase string
        if rp == True:
            t = [c for c in t if not (c in punct)] # non-punct characters

        # Keep punctuation
        else:
            new_arr = []
            for c in t:
                if not (c in punct):
                    new_arr.append(c)
                else:
                    new_arr.extend([' ', c, ' '])
            t = new_arr
        t = ''.join(t) # convert back to string
        words = t.strip().split() # tokenizes
        for i in range(len(words)-1):
            b = (words[i], words[i+1])
            if b in bigramSet:
                idfCounter[b] += 1
    return idfCounter

# Calculate tf-idf scores for unigram or bigram along with punctuation remove or preserve
def tf_idf7(gram, datum, data, ub, rp, idfCounter):
    counter = 0
    t = datum['text']
    t = t.lower() # lowercase string
    if rp == True:
        t = [c for c in t if not (c in punct)] # non-punct characters

    # Keep punctuation
    else:
        new_arr = []
        for c in t:
            if not (c in punct):
                new_arr.append(c)
            else:
                new_arr.extend([' ', c, ' '])
        t = new_arr
    t = ''.join(t) # convert back to string
    words = t.strip().split() # tokenizes

    if ub == True:
        for w in words:
            if w == gram:
                counter += 1
    else:
        for i in range(len(words)-1):
            b = (words[i], words[i+1])
            if b == gram:
                counter += 1
    return counter * np.log10(len(data) / idfCounter[gram])
```

[651]:

```
# Creates word feature for the feature matrix
def feature7(datum, Id, Set, ub, rp, tc, idfCounter):


    feat = [0]*(len(Set))

    t = datum['text']
    t = t.lower() # lowercase string
     # Remove punctuation
    if rp == True:
        t = [c for c in t if not (c in punct)] # non-punct characters

    # Keep punctuation
    else:
        new_arr = []
        for c in t:
            if not (c in punct):
                new_arr.append(c)
            else:
                new_arr.extend([' ', c, ' '])
        t = new_arr
    t = ''.join(t) # convert back to string
    words = t.strip().split() # tokenizes

    # unigrams
    if ub == True:
        for w in words:
            if not (w in Set): continue

            # True = tfidf
            if tc == True:
                feat[Id[w]] = tf_idf7(w, datum, training_data, ub, rp, idfCounter)
            # False = count
            else:
                feat[Id[w]] += 1
    # bigrams
    else:
        for i in range(len(words)-1):
            b = (words[i], words[i+1])
            if not (b in Set): continue
            # True = tfidf
            if tc == True:
                feat[Id[b]] = tf_idf7(b, datum, training_data, ub, rp, idfCounter)
            # False = count
            else:
                feat[Id[b]] += 1

    feat.append(1)
    return feat

# Gets the dictionary set for top 1000 unigrams, bigrams with punctation/no punctuation
def q7Set(ub, rp, tc):

    # False = bigram
    if ub == False:
        BigramCount = defaultdict(int)
        totalBigrams = 0

        for d in training_data:
            t = d['text']
            t = t.lower() # lowercase string

            # Remove punctuation
            if rp == True:
                t = [c for c in t if not (c in punct)] # non-punct characters

            # Keep punctuation
            else:
                new_arr = []
                for c in t:
                    if not (c in punct):
                        new_arr.append(c)
                    else:
                        new_arr.extend([' ', c, ' '])
                t = new_arr
            t = ''.join(t) # convert back to string
            words = t.strip().split() # tokenizes
            for i in range(len(words)-1):
                totalBigrams += 1
                BigramCount[(words[i], words[i+1])] += 1

        counts = [(BigramCount[b], b) for b in BigramCount]
        counts.sort()
        counts.reverse()

        bigrams = [b[1] for b in counts[:1000]]
        bigramId = dict(zip(bigrams, range(len(bigrams))))
        bigramSet = set(bigrams)


        return bigrams, bigramId, bigramSet




    else:

        unigramCount = defaultdict(int)
        totalUnigrams = 0

        for d in training_data:
            t = d['text']
            t = t.lower() # lowercase string
           # Remove punctuation
            if rp == True:
                t = [c for c in t if not (c in punct)] # non-punct characters

            # Keep punctuation
            else:
                new_arr = []
                for c in t:
                    if not (c in punct):
                        new_arr.append(c)
                    else:
                        new_arr.extend([' ', c, ' '])
                t = new_arr
            t = ''.join(t) # convert back to string
            words = t.strip().split() # tokenizes
            for w in words:
                totalUnigrams += 1
                unigramCount[w] += 1

        countsUni = [(unigramCount[w], w) for w in unigramCount]
        countsUni.sort()
        countsUni.reverse()

        unigrams = [w[1] for w in countsUni[:1000]]
        unigramId = dict(zip(unigrams, range(len(unigrams))))
        unigramSet = set(unigrams)

        return unigrams, unigramId, unigramSet
```

[707]:

```
UB_Map = {True: 'Unigram', False: 'Bigram'}
RP_Map = {True: 'Remove Punctuation', False: 'Preserve Punctuation'}
TC_Map = {True: 'Tf-Idf Scores', False: 'Word Counts'}

Cs = [0.01, 0.1, 1, 10, 100]

# Lists to constuct performance table
C_df = []

uni_df = []
bi_df = []

rp_df = []
pp_df = []

tfidf_df = []
wc_df = []

val_mse_df = []
test_mse_df = []
```

[708]:

```
# Unigrams, Remove Punctuation, tfidf scores

grams, gramId, gramSet = q7Set(True, True, True)

IdfCounter = getIdfMapUnigram(gramSet, training_data, True)

X_train = [feature7(d, gramId, gramSet, True, True, True, IdfCounter) for d in␣
 ↪training_data]
y_train = [np.log2(d['hours'] + 1) for d in training_data]
```

```
X_validation = [feature7(d, gramId, gramSet, True, True, True, IdfCounter) for␣
 ↪d in validation_data]
y_validation = [np.log2(d['hours'] + 1) for d in validation_data]

X_test = [feature7(d, gramId, gramSet, True, True, True, IdfCounter) for d in␣
 ↪test_data]
y_test = [np.log2(d['hours'] + 1) for d in test_data]


for C in Cs:
    clf = linear_model.Ridge(C, fit_intercept=False)
    clf.fit(X_train, y_train)
    theta = clf.coef_

    predictions = clf.predict(X_validation)
    MSE = sum((predictions - y_validation)**2) / len(y_validation)
    print('MSE Validation C =', C,'(' + UB_Map[True] + ', ' + RP_Map[True] + ',␣
 ↪' + TC_Map[True] + ') =', MSE)

    val_mse_df.append(MSE)

    predictions = clf.predict(X_test)
    MSE = sum((predictions - y_test)**2) / len(y_test)
    print('MSE Test C =', C,'(' + UB_Map[True] + ', ' + RP_Map[True] + ', ' +␣
 ↪TC_Map[True] + ') =', MSE)

    test_mse_df.append(MSE)

    print()

    C_df.append(C)
    uni_df.append(True)
    bi_df.append(False)

    rp_df.append(True)
    pp_df.append(False)

    tfidf_df.append(True)
    wc_df.append(False)
```

```
MSE Validation C = 0.01 (Unigram, Remove Punctuation, Tf-Idf Scores) =
5.673806372335936
MSE Test C = 0.01 (Unigram, Remove Punctuation, Tf-Idf Scores) =
5.44341643454938

MSE Validation C = 0.1 (Unigram, Remove Punctuation, Tf-Idf Scores) =
5.664963930672582
MSE Test C = 0.1 (Unigram, Remove Punctuation, Tf-Idf Scores) =
5.442522596772181

MSE Validation C = 1 (Unigram, Remove Punctuation, Tf-Idf Scores) =
5.644370081585309
MSE Test C = 1 (Unigram, Remove Punctuation, Tf-Idf Scores) = 5.442046784564783

MSE Validation C = 10 (Unigram, Remove Punctuation, Tf-Idf Scores) =
5.5789306327577215
MSE Test C = 10 (Unigram, Remove Punctuation, Tf-Idf Scores) = 5.397073383016831

MSE Validation C = 100 (Unigram, Remove Punctuation, Tf-Idf Scores) =
5.2129695143827535
MSE Test C = 100 (Unigram, Remove Punctuation, Tf-Idf Scores) =
5.122243582557432
```

[709]:

```
# Bigrams, Remove puncutation, tfidf scores
grams, gramId, gramSet = q7Set(False, True, True)

IdfCounter = getIdfMapBigram(gramSet, training_data, True)

X_train = [feature7(d, gramId, gramSet, False, True, True, IdfCounter) for d in␣
 ↪training_data]
y_train = [np.log2(d['hours'] + 1) for d in training_data]

X_validation = [feature7(d, gramId, gramSet, False, True, True, IdfCounter) for␣
 ↪d in validation_data]
y_validation = [np.log2(d['hours'] + 1) for d in validation_data]

X_test = [feature7(d, gramId, gramSet, False, True, True, IdfCounter) for d in␣
 ↪test_data]
y_test = [np.log2(d['hours'] + 1) for d in test_data]


for C in Cs:
    clf = linear_model.Ridge(C, fit_intercept=False)
    clf.fit(X_train, y_train)
    theta = clf.coef_

    predictions = clf.predict(X_validation)
    MSE = sum((predictions - y_validation)**2) / len(y_validation)
    print('MSE Validation C =', C,'(' + UB_Map[False] + ', ' + RP_Map[True] +␣
↪', ' + TC_Map[True] + ') =', MSE)

    val_mse_df.append(MSE)

    predictions = clf.predict(X_test)
    MSE = sum((predictions - y_test)**2) / len(y_test)
    print('MSE Test C =', C,'(' + UB_Map[False] + ', ' + RP_Map[True] + ', ' +␣
↪TC_Map[True] + ') =', MSE)

    test_mse_df.append(MSE)

    print()

    C_df.append(C)
    uni_df.append(False)
    bi_df.append(True)

    rp_df.append(True)
    pp_df.append(False)

    tfidf_df.append(True)
    wc_df.append(False)
```

```
MSE Validation C = 0.01 (Bigram, Remove Punctuation, Tf-Idf Scores) =
5.673287403837088
MSE Test C = 0.01 (Bigram, Remove Punctuation, Tf-Idf Scores) =
5.506209233571954

MSE Validation C = 0.1 (Bigram, Remove Punctuation, Tf-Idf Scores) =
5.672797087486668
MSE Test C = 0.1 (Bigram, Remove Punctuation, Tf-Idf Scores) = 5.505538566595185

MSE Validation C = 1 (Bigram, Remove Punctuation, Tf-Idf Scores) =
5.668004372267518
MSE Test C = 1 (Bigram, Remove Punctuation, Tf-Idf Scores) = 5.499240629174145

MSE Validation C = 10 (Bigram, Remove Punctuation, Tf-Idf Scores) =
5.621993571271137
MSE Test C = 10 (Bigram, Remove Punctuation, Tf-Idf Scores) = 5.448469226938605

MSE Validation C = 100 (Bigram, Remove Punctuation, Tf-Idf Scores) =
5.366705958094359
MSE Test C = 100 (Bigram, Remove Punctuation, Tf-Idf Scores) = 5.215462371009729
```

[710]:

```
# Unigrams, Preserve Punctuation, tfidf scores
grams, gramId, gramSet = q7Set(True, False, True)

IdfCounter = getIdfMapUnigram(gramSet, training_data, False)

X_train = [feature7(d, gramId, gramSet, True, False, True, IdfCounter) for d in training_data]
y_train = [np.log2(d['hours'] + 1) for d in training_data]

X_validation = [feature7(d, gramId, gramSet, True, False, True, IdfCounter) for d in validation_data]
y_validation = [np.log2(d['hours'] + 1) for d in validation_data]

X_test = [feature7(d, gramId, gramSet, True, False, True, IdfCounter) for d in test_data]
y_test = [np.log2(d['hours'] + 1) for d in test_data]


for C in Cs:
    clf = linear_model.Ridge(C, fit_intercept=False)
    clf.fit(X_train, y_train)
    theta = clf.coef_

    predictions = clf.predict(X_validation)
    MSE = sum((predictions - y_validation)**2) / len(y_validation)
    print('MSE Validation C =', C,'(' + UB_Map[True] + ', ' + RP_Map[False] + ', ' + TC_Map[True] + ') =', MSE)

    val_mse_df.append(MSE)

    predictions = clf.predict(X_test)
    MSE = sum((predictions - y_test)**2) / len(y_test)
    print('MSE Test C =', C,'(' + UB_Map[True] + ', ' + RP_Map[False] + ', ' + TC_Map[True] + ') =', MSE)

    test_mse_df.append(MSE)

    print()

    C_df.append(C)
    uni_df.append(True)
    bi_df.append(False)

    rp_df.append(False)
    pp_df.append(True)

    tfidf_df.append(True)
    wc_df.append(False)
```

```
MSE Validation C = 0.01 (Unigram, Preserve Punctuation, Tf-Idf Scores) =
5.570218397178122
MSE Test C = 0.01 (Unigram, Preserve Punctuation, Tf-Idf Scores) =
5.422839226112137

MSE Validation C = 0.1 (Unigram, Preserve Punctuation, Tf-Idf Scores) =
5.568655202923451
MSE Test C = 0.1 (Unigram, Preserve Punctuation, Tf-Idf Scores) =
5.4211872568211925

MSE Validation C = 1 (Unigram, Preserve Punctuation, Tf-Idf Scores) =
5.567519236384305
MSE Test C = 1 (Unigram, Preserve Punctuation, Tf-Idf Scores) =
5.419563281432396

MSE Validation C = 10 (Unigram, Preserve Punctuation, Tf-Idf Scores) =
5.517231538305895
MSE Test C = 10 (Unigram, Preserve Punctuation, Tf-Idf Scores) =
5.376351913572079

MSE Validation C = 100 (Unigram, Preserve Punctuation, Tf-Idf Scores) =
5.202233460893193
MSE Test C = 100 (Unigram, Preserve Punctuation, Tf-Idf Scores) =
5.11910178254146
```

[711]:

```
# Bigrams, Preserve Punctuation, tfidf scores
grams, gramId, gramSet = q7Set(False, False, True)

IdfCounter = getIdfMapBigram(gramSet, training_data, False)

X_train = [feature7(d, gramId, gramSet, False, False, True, IdfCounter) for d␣
 ↪in training_data]
y_train = [np.log2(d['hours'] + 1) for d in training_data]

X_validation = [feature7(d, gramId, gramSet, False, False, True, IdfCounter)␣
 ↪for d in validation_data]
y_validation = [np.log2(d['hours'] + 1) for d in validation_data]

X_test = [feature7(d, gramId, gramSet, False, False, True, IdfCounter) for d in␣
 ↪test_data]
y_test = [np.log2(d['hours'] + 1) for d in test_data]


for C in Cs:
    clf = linear_model.Ridge(C, fit_intercept=False)
    clf.fit(X_train, y_train)
    theta = clf.coef_

    predictions = clf.predict(X_validation)
    MSE = sum((predictions - y_validation)**2) / len(y_validation)
    print('MSE Validation C =', C,'(' + UB_Map[False] + ', ' + RP_Map[False] +␣
↪', ' + TC_Map[True] + ') =', MSE)

    val_mse_df.append(MSE)

    predictions = clf.predict(X_test)
    MSE = sum((predictions - y_test)**2) / len(y_test)
    print('MSE Test C =', C,'(' + UB_Map[False] + ', ' + RP_Map[False] + ', ' +␣
↪TC_Map[True] + ') =', MSE)

    test_mse_df.append(MSE)

    print()

    C_df.append(C)
    uni_df.append(False)
    bi_df.append(True)

    rp_df.append(False)
    pp_df.append(True)

    tfidf_df.append(True)
    wc_df.append(False)
```

```
MSE Validation C = 0.01 (Bigram, Preserve Punctuation, Tf-Idf Scores) =
5.582117883160818
MSE Test C = 0.01 (Bigram, Preserve Punctuation, Tf-Idf Scores) =
5.525768841253606

MSE Validation C = 0.1 (Bigram, Preserve Punctuation, Tf-Idf Scores) =
5.5811412448533835
MSE Test C = 0.1 (Bigram, Preserve Punctuation, Tf-Idf Scores) =
5.524634175402464

MSE Validation C = 1 (Bigram, Preserve Punctuation, Tf-Idf Scores) =
5.574056215986605
MSE Test C = 1 (Bigram, Preserve Punctuation, Tf-Idf Scores) = 5.515303428263916

MSE Validation C = 10 (Bigram, Preserve Punctuation, Tf-Idf Scores) =
5.53675784403914
MSE Test C = 10 (Bigram, Preserve Punctuation, Tf-Idf Scores) =
5.461732251629773

MSE Validation C = 100 (Bigram, Preserve Punctuation, Tf-Idf Scores) =
5.272388170686014
MSE Test C = 100 (Bigram, Preserve Punctuation, Tf-Idf Scores) =
5.232844454976385
```

[712]:

```
# Unigrams, Remove Punctuation, word counts
grams, gramId, gramSet = q7Set(True, True, False)

IdfCounter = getIdfMapUnigram(gramSet, training_data, True)

X_train = [feature7(d, gramId, gramSet, True, True, False, IdfCounter) for d in␣
 ↪training_data]
y_train = [np.log2(d['hours'] + 1) for d in training_data]

X_validation = [feature7(d, gramId, gramSet, True, True, False, IdfCounter) for␣
 ↪d in validation_data]
y_validation = [np.log2(d['hours'] + 1) for d in validation_data]

X_test = [feature7(d, gramId, gramSet, True, True, False, IdfCounter) for d in␣
 ↪test_data]
y_test = [np.log2(d['hours'] + 1) for d in test_data]


for C in Cs:
    clf = linear_model.Ridge(C, fit_intercept=False)
    clf.fit(X_train, y_train)
    theta = clf.coef_

    predictions = clf.predict(X_validation)
    MSE = sum((predictions - y_validation)**2) / len(y_validation)
    print('MSE Validation C =', C,'(' + UB_Map[True] + ', ' + RP_Map[True] + ',␣
 ↪' + TC_Map[False] + ') =', MSE)

    val_mse_df.append(MSE)

    predictions = clf.predict(X_test)
    MSE = sum((predictions - y_test)**2) / len(y_test)
    print('MSE Test C =', C,'(' + UB_Map[True] + ', ' + RP_Map[True] + ', ' +␣
 ↪TC_Map[False] + ') =', MSE)

    test_mse_df.append(MSE)

    print()
    C_df.append(C)
    uni_df.append(True)
    bi_df.append(False)
    rp_df.append(True)
    pp_df.append(False)
    tfidf_df.append(False)
    wc_df.append(True)
```

```
MSE Validation C = 0.01 (Unigram, Remove Punctuation, Word Counts) =
5.674954707791254
MSE Test C = 0.01 (Unigram, Remove Punctuation, Word Counts) = 5.443446979850613

MSE Validation C = 0.1 (Unigram, Remove Punctuation, Word Counts) =
5.672776933245138
MSE Test C = 0.1 (Unigram, Remove Punctuation, Word Counts) = 5.44173383018089

MSE Validation C = 1 (Unigram, Remove Punctuation, Word Counts) =
5.65154026758534
MSE Test C = 1 (Unigram, Remove Punctuation, Word Counts) = 5.425086121336876

MSE Validation C = 10 (Unigram, Remove Punctuation, Word Counts) =
5.481431537033336
MSE Test C = 10 (Unigram, Remove Punctuation, Word Counts) = 5.295566510408608

MSE Validation C = 100 (Unigram, Remove Punctuation, Word Counts) =
4.968279141466649
MSE Test C = 100 (Unigram, Remove Punctuation, Word Counts) = 4.949465612094928
```

[713]:

```
# Bigrams, Remove Punctuation, word counts
grams, gramId, gramSet = q7Set(False, True, False)

IdfCounter = getIdfMapBigram(gramSet, training_data, True)

X_train = [feature7(d, gramId, gramSet, False, True, False, IdfCounter) for d␣
 ↪in training_data]
y_train = [np.log2(d['hours'] + 1) for d in training_data]

X_validation = [feature7(d, gramId, gramSet, False, True, False, IdfCounter)␣
 ↪for d in validation_data]
y_validation = [np.log2(d['hours'] + 1) for d in validation_data]

X_test = [feature7(d, gramId, gramSet, False, True, False, IdfCounter) for d in␣
 ↪test_data]
y_test = [np.log2(d['hours'] + 1) for d in test_data]
```

```
for C in Cs:
    clf = linear_model.Ridge(C, fit_intercept=False)
    clf.fit(X_train, y_train)
    theta = clf.coef_

    predictions = clf.predict(X_validation)
    MSE = sum((predictions - y_validation)**2) / len(y_validation)
    print('MSE Validation C =', C,'(' + UB_Map[False] + ', ' + RP_Map[True] + ', ' + TC_Map[False] + ') =', MSE)

    val_mse_df.append(MSE)

    predictions = clf.predict(X_test)
    MSE = sum((predictions - y_test)**2) / len(y_test)
    print('MSE Test C =', C,'(' + UB_Map[False] + ', ' + RP_Map[True] + ', ' + TC_Map[False] + ') =', MSE)

    test_mse_df.append(MSE)

    print()

    C_df.append(C)
    uni_df.append(False)
    bi_df.append(True)

    rp_df.append(True)
    pp_df.append(False)

    tfidf_df.append(False)
    wc_df.append(True)
```

```
MSE Validation C = 0.01 (Bigram, Remove Punctuation, Word Counts) =
5.67317280728575
MSE Test C = 0.01 (Bigram, Remove Punctuation, Word Counts) = 5.505991995034113

MSE Validation C = 0.1 (Bigram, Remove Punctuation, Word Counts) =
5.671681325319052
MSE Test C = 0.1 (Bigram, Remove Punctuation, Word Counts) = 5.503457637327854

MSE Validation C = 1 (Bigram, Remove Punctuation, Word Counts) =
5.657019481153744
MSE Test C = 1 (Bigram, Remove Punctuation, Word Counts) = 5.481632097339202

MSE Validation C = 10 (Bigram, Remove Punctuation, Word Counts) =
5.541543954145307
MSE Test C = 10 (Bigram, Remove Punctuation, Word Counts) = 5.344781053758389
MSE Validation C = 100 (Bigram, Remove Punctuation, Word Counts) =
5.287558047852009
MSE Test C = 100 (Bigram, Remove Punctuation, Word Counts) = 5.08111846367742
```

[714]:

```
# Unigrams, Preserve Punctuation, word counts
grams, gramId, gramSet = q7Set(True, False, False)

IdfCounter = getIdfMapUnigram(gramSet, training_data, False)

X_train = [feature7(d, gramId, gramSet, True, False, False, IdfCounter) for d␣
 ↪in training_data]
y_train = [np.log2(d['hours'] + 1) for d in training_data]

X_validation = [feature7(d, gramId, gramSet, True, False, False, IdfCounter)␣
 ↪for d in validation_data]
y_validation = [np.log2(d['hours'] + 1) for d in validation_data]

X_test = [feature7(d, gramId, gramSet, True, False, False, IdfCounter) for d in␣
 ↪test_data]
y_test = [np.log2(d['hours'] + 1) for d in test_data]


for C in Cs:
    clf = linear_model.Ridge(C, fit_intercept=False)
    clf.fit(X_train, y_train)
    theta = clf.coef_

    predictions = clf.predict(X_validation)
    MSE = sum((predictions - y_validation)**2) / len(y_validation)
    print('MSE Validation C =', C,'(' + UB_Map[True] + ', ' + RP_Map[False] +␣
 ↪', ' + TC_Map[False] + ') =', MSE)

    val_mse_df.append(MSE)

    predictions = clf.predict(X_test)
    MSE = sum((predictions - y_test)**2) / len(y_test)
    print('MSE Test C =', C,'(' + UB_Map[True] + ', ' + RP_Map[False] + ', ' +␣
 ↪TC_Map[False] + ') =', MSE)

    test_mse_df.append(MSE)

    print()

    C_df.append(C)
    uni_df.append(True)
    bi_df.append(False)
    rp_df.append(False)
    pp_df.append(True)

    tfidf_df.append(False)
    wc_df.append(True)
```

```
MSE Validation C = 0.01 (Unigram, Preserve Punctuation, Word Counts) =
5.570524252790048
MSE Test C = 0.01 (Unigram, Preserve Punctuation, Word Counts) =
5.423149116056916

MSE Validation C = 0.1 (Unigram, Preserve Punctuation, Word Counts) =
5.568800507111982
MSE Test C = 0.1 (Unigram, Preserve Punctuation, Word Counts) =
5.421671333791256

MSE Validation C = 1 (Unigram, Preserve Punctuation, Word Counts) =
5.551952768534205
MSE Test C = 1 (Unigram, Preserve Punctuation, Word Counts) = 5.407268381152867

MSE Validation C = 10 (Unigram, Preserve Punctuation, Word Counts) =
5.41471873855819
MSE Test C = 10 (Unigram, Preserve Punctuation, Word Counts) = 5.292599618963542

MSE Validation C = 100 (Unigram, Preserve Punctuation, Word Counts) =
4.978360758164593
MSE Test C = 100 (Unigram, Preserve Punctuation, Word Counts) = 4.96190152372311
```

[715]:

```
# Bigrams, Preserve Punctuation, word counts
grams, gramId, gramSet = q7Set(False, False, False)

IdfCounter = getIdfMapBigram(gramSet, training_data, False)

X_train = [feature7(d, gramId, gramSet, False, False, False, IdfCounter) for d␣
 ↪in training_data]
y_train = [np.log2(d['hours'] + 1) for d in training_data]

X_validation = [feature7(d, gramId, gramSet, False, False, False, IdfCounter)␣
 ↪for d in validation_data]
y_validation = [np.log2(d['hours'] + 1) for d in validation_data]

X_test = [feature7(d, gramId, gramSet, False, False, False, IdfCounter) for d␣
 ↪in test_data]
y_test = [np.log2(d['hours'] + 1) for d in test_data]
```

```
for C in Cs:
    clf = linear_model.Ridge(C, fit_intercept=False)
    clf.fit(X_train, y_train)
    theta = clf.coef_

    predictions = clf.predict(X_validation)
    MSE = sum((predictions - y_validation)**2) / len(y_validation)
    print('MSE Validation C =', C,'(' + UB_Map[False] + ', ' + RP_Map[False] +␣
 ↪', ' + TC_Map[False] + ') =', MSE)

    val_mse_df.append(MSE)

    predictions = clf.predict(X_test)
    MSE = sum((predictions - y_test)**2) / len(y_test)
    print('MSE Test C =', C,'(' + UB_Map[False] + ', ' + RP_Map[False] + ', ' +␣
 ↪TC_Map[False] + ') =', MSE)

    test_mse_df.append(MSE)

    print()

    C_df.append(C)
    uni_df.append(False)
    bi_df.append(True)

    rp_df.append(False)
    pp_df.append(True)

    tfidf_df.append(False)
    wc_df.append(True)
```

```
MSE Validation C = 0.01 (Bigram, Preserve Punctuation, Word Counts) =
5.581761417866405
MSE Test C = 0.01 (Bigram, Preserve Punctuation, Word Counts) =
5.525502739262317

MSE Validation C = 0.1 (Bigram, Preserve Punctuation, Word Counts) =
5.5779367383972795
MSE Test C = 0.1 (Bigram, Preserve Punctuation, Word Counts) = 5.522169218695633

MSE Validation C = 1 (Bigram, Preserve Punctuation, Word Counts) =
5.553447642596715
MSE Test C = 1 (Bigram, Preserve Punctuation, Word Counts) = 5.498788372251529
MSE Validation C = 10 (Bigram, Preserve Punctuation, Word Counts) =
5.40154892954481
MSE Test C = 10 (Bigram, Preserve Punctuation, Word Counts) = 5.375797766715348

MSE Validation C = 100 (Bigram, Preserve Punctuation, Word Counts) =
5.064762706947522
MSE Test C = 100 (Bigram, Preserve Punctuation, Word Counts) = 5.103291923113969
```

[718]:

```
des_df = []
for i in range(40):
    s = ""
    if uni_df[i]:
        s += "Unigram, "
    else:
        s += "Bigram, "

    if rp_df[i]:
        s += "Remove Punctuation, "
    else:
        s += "Preserve Punctuation, "

    if tfidf_df[i]:
        s += "Tf-Idf Scores"
    else:
        s += "Word Counts"

    des_df.append(s)


d = {
     'Model Description': des_df,
     'Regularization Parameter': C_df,
     'Validation MSE': val_mse_df,
     'Test MSE': test_mse_df
    }
model_num = ['Model ' + str(n) for n in range(1,41)]
performance_table = pd.DataFrame(data=d, index=model_num, columns=['Model␣
 ↪Description', 'Regularization Parameter', 'Validation MSE', 'Test MSE'])
print('Performance Table')
performance_table
```

```
Performance Table
```

[718]:

```
                                         Model Description  \
Model 1     Unigram, Remove Punctuation, Tf-Idf Scores
Model 2     Unigram, Remove Punctuation, Tf-Idf Scores
Model 3     Unigram, Remove Punctuation, Tf-Idf Scores
Model 4     Unigram, Remove Punctuation, Tf-Idf Scores
Model 5     Unigram, Remove Punctuation, Tf-Idf Scores
Model 6      Bigram, Remove Punctuation, Tf-Idf Scores
Model 7      Bigram, Remove Punctuation, Tf-Idf Scores
Model 8      Bigram, Remove Punctuation, Tf-Idf Scores
Model 9      Bigram, Remove Punctuation, Tf-Idf Scores
Model 10     Bigram, Remove Punctuation, Tf-Idf Scores
Model 11  Unigram, Preserve Punctuation, Tf-Idf Scores
Model 12  Unigram, Preserve Punctuation, Tf-Idf Scores
Model 13  Unigram, Preserve Punctuation, Tf-Idf Scores
Model 14  Unigram, Preserve Punctuation, Tf-Idf Scores
Model 15  Unigram, Preserve Punctuation, Tf-Idf Scores
Model 16   Bigram, Preserve Punctuation, Tf-Idf Scores
Model 17   Bigram, Preserve Punctuation, Tf-Idf Scores
Model 18   Bigram, Preserve Punctuation, Tf-Idf Scores
Model 19   Bigram, Preserve Punctuation, Tf-Idf Scores
Model 20   Bigram, Preserve Punctuation, Tf-Idf Scores
Model 21      Unigram, Remove Punctuation, Word Counts
Model 22      Unigram, Remove Punctuation, Word Counts
Model 23      Unigram, Remove Punctuation, Word Counts
Model 24      Unigram, Remove Punctuation, Word Counts
Model 25      Unigram, Remove Punctuation, Word Counts
Model 26       Bigram, Remove Punctuation, Word Counts
Model 27       Bigram, Remove Punctuation, Word Counts
Model 28       Bigram, Remove Punctuation, Word Counts
Model 29       Bigram, Remove Punctuation, Word Counts
Model 30       Bigram, Remove Punctuation, Word Counts
Model 31    Unigram, Preserve Punctuation, Word Counts
Model 32    Unigram, Preserve Punctuation, Word Counts
Model 33    Unigram, Preserve Punctuation, Word Counts
Model 34    Unigram, Preserve Punctuation, Word Counts
Model 35    Unigram, Preserve Punctuation, Word Counts
Model 36     Bigram, Preserve Punctuation, Word Counts
Model 37     Bigram, Preserve Punctuation, Word Counts
Model 38     Bigram, Preserve Punctuation, Word Counts
Model 39     Bigram, Preserve Punctuation, Word Counts
Model 40     Bigram, Preserve Punctuation, Word Counts

          Regularization Parameter  Validation MSE  Test MSE
Model 1                       0.01        5.673806  5.443416
Model 2                       0.10        5.664964  5.442523
Model 3                       1.00        5.644370  5.442047
Model 4                      10.00        5.578931  5.397073
Model 5                     100.00        5.212970  5.122244
Model 6                       0.01        5.673287  5.506209
Model 7                       0.10        5.672797  5.505539
Model 8                       1.00        5.668004  5.499241
Model 9                      10.00        5.621994  5.448469
Model 10                    100.00        5.366706  5.215462
Model 11                      0.01        5.570218  5.422839
Model 12                      0.10        5.568655  5.421187
Model 13                      1.00        5.567519  5.419563
Model 14                     10.00        5.517232  5.376352
Model 15                    100.00        5.202233  5.119102
Model 16                      0.01        5.582118  5.525769
Model 17                      0.10        5.581141  5.524634
Model 18                      1.00        5.574056  5.515303
Model 19                     10.00        5.536758  5.461732
Model 20                    100.00        5.272388  5.232844
Model 21                      0.01        5.674955  5.443447
Model 22                      0.10        5.672777  5.441734
Model 23                      1.00        5.651540  5.425086
Model 24                     10.00        5.481432  5.295567
Model 25                    100.00        4.968279  4.949466
Model 26                      0.01        5.673173  5.505992
Model 27                      0.10        5.671681  5.503458
Model 28                      1.00        5.657019  5.481632
Model 29                     10.00        5.541544  5.344781
Model 30                    100.00        5.287558  5.081118
Model 31                      0.01        5.570524  5.423149
Model 32                      0.10        5.568801  5.421671
Model 33                      1.00        5.551953  5.407268
Model 34                     10.00        5.414719  5.292600
Model 35                    100.00        4.978361  4.961902
Model 36                      0.01        5.581761  5.525503
Model 37                      0.10        5.577937  5.522169
Model 38                      1.00        5.553448  5.498788
Model 39                     10.00        5.401549  5.375798
Model 40                    100.00        5.064763  5.103292
```

```
[ ]:
```

```
[719]: print('Performance Table sorted by Validation MSE')
       performance_table.sort_values('Validation MSE')
```

```
Performance Table sorted by Validation MSE
```

```
[719]:
                                     Model Description  \
Model 25      Unigram, Remove Punctuation, Word Counts
Model 35    Unigram, Preserve Punctuation, Word Counts
Model 40     Bigram, Preserve Punctuation, Word Counts
Model 15  Unigram, Preserve Punctuation, Tf-Idf Scores
Model 5     Unigram, Remove Punctuation, Tf-Idf Scores
Model 20   Bigram, Preserve Punctuation, Tf-Idf Scores
Model 30       Bigram, Remove Punctuation, Word Counts
Model 10     Bigram, Remove Punctuation, Tf-Idf Scores
Model 39     Bigram, Preserve Punctuation, Word Counts
Model 34    Unigram, Preserve Punctuation, Word Counts
Model 24      Unigram, Remove Punctuation, Word Counts
Model 14  Unigram, Preserve Punctuation, Tf-Idf Scores
Model 19   Bigram, Preserve Punctuation, Tf-Idf Scores
Model 29       Bigram, Remove Punctuation, Word Counts
Model 33    Unigram, Preserve Punctuation, Word Counts
Model 38     Bigram, Preserve Punctuation, Word Counts
Model 13  Unigram, Preserve Punctuation, Tf-Idf Scores
Model 12  Unigram, Preserve Punctuation, Tf-Idf Scores
Model 32    Unigram, Preserve Punctuation, Word Counts
Model 11  Unigram, Preserve Punctuation, Tf-Idf Scores
Model 31    Unigram, Preserve Punctuation, Word Counts
Model 18   Bigram, Preserve Punctuation, Tf-Idf Scores
Model 37     Bigram, Preserve Punctuation, Word Counts
Model 4     Unigram, Remove Punctuation, Tf-Idf Scores
Model 17   Bigram, Preserve Punctuation, Tf-Idf Scores
Model 36     Bigram, Preserve Punctuation, Word Counts
Model 16   Bigram, Preserve Punctuation, Tf-Idf Scores
Model 9      Bigram, Remove Punctuation, Tf-Idf Scores
Model 3     Unigram, Remove Punctuation, Tf-Idf Scores
Model 23      Unigram, Remove Punctuation, Word Counts
Model 28       Bigram, Remove Punctuation, Word Counts
Model 2     Unigram, Remove Punctuation, Tf-Idf Scores
Model 8      Bigram, Remove Punctuation, Tf-Idf Scores
Model 27       Bigram, Remove Punctuation, Word Counts
Model 22      Unigram, Remove Punctuation, Word Counts
Model 7      Bigram, Remove Punctuation, Tf-Idf Scores
Model 26       Bigram, Remove Punctuation, Word Counts
Model 6      Bigram, Remove Punctuation, Tf-Idf Scores
Model 1     Unigram, Remove Punctuation, Tf-Idf Scores
Model 21      Unigram, Remove Punctuation, Word Counts

          Regularization Parameter  Validation MSE  Test MSE
Model 25                    100.00        4.968279  4.949466
Model 35                    100.00        4.978361  4.961902
Model 40                    100.00        5.064763  5.103292
Model 15                    100.00        5.202233  5.119102
Model 5                     100.00        5.212970  5.122244
Model 20                    100.00        5.272388  5.232844
Model 30                    100.00        5.287558  5.081118
Model 10                    100.00        5.366706  5.215462
Model 39                     10.00        5.401549  5.375798
Model 34                     10.00        5.414719  5.292600
Model 24                    10.00        5.481432  5.295567
Model 14                    10.00        5.517232  5.376352
Model 19                    10.00        5.536758  5.461732
Model 29                    10.00        5.541544  5.344781
Model 33                     1.00        5.551953  5.407268
Model 38                     1.00        5.553448  5.498788
Model 13                     1.00        5.567519  5.419563
Model 12                     0.10        5.568655  5.421187
Model 32                     0.10        5.568801  5.421671
Model 11                     0.01        5.570218  5.422839
Model 31                     0.01        5.570524  5.423149
Model 18                     1.00        5.574056  5.515303
Model 37                     0.10        5.577937  5.522169
Model 4                     10.00        5.578931  5.397073
Model 17                     0.10        5.581141  5.524634
Model 36                     0.01        5.581761  5.525503
Model 16                     0.01        5.582118  5.525769
Model 9                     10.00        5.621994  5.448469
Model 3                      1.00        5.644370  5.442047
Model 23                     1.00        5.651540  5.425086
Model 28                     1.00        5.657019  5.481632
Model 2                      0.10        5.664964  5.442523
Model 8                      1.00        5.668004  5.499241
Model 27                     0.10        5.671681  5.503458
Model 22                     0.10        5.672777  5.441734
Model 7                      0.10        5.672797  5.505539
Model 26                     0.01        5.673173  5.505992
Model 6                      0.01        5.673287  5.506209
Model 1                      0.01        5.673806  5.443416
Model 21                     0.01        5.674955  5.443447
```

After sorting the performance table by Validation MSE, the best performing model for this randomized train, validation, test split was Model 25 (Unigram, Remove Punctuation, Word Counts) because it had the smallest validation MSE compared to the other models. From the table we can also see that the top 8 models with the lowest validation MSEs are all of the different variation types (8 variations) where the regularization parameter was 100.

From our best performing model, we can see that the corresponding Test MSE for the best model (Model 25) is 4.949466

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```