## CSEE 5590 0001- Python and Deep Learning

## Fall 2018

## Python Lab Assignment 2

## Submitted On 10/10/2018

## Name: Alexandria Piatt and Alexander Larios

## Class IDs (respectively): 23 15

## TABLE OF CONTENTS

## 1. Author

## 2. Objective

## 3. Features

## 4. Configuration

5. Input/Output Screenshots

6. Implementation & Full Source Code

7. References

**AUTHORS**

This lab report corresponds to the second lab assignment in CS 5590 0001 Python and Deep-Learning. ALEXANDRIA PIATT (ID: 23) and ALEXANDER LARIOS (ID: 15) are both in their last year at UMKC, both are completing a B.S. in C.S. in December 2018. The leader of the course is SARIA GOUDARZVAND.

**OBJECTIVE**

By completing the tasks for this lab we were able to practice simple deep learning skills.

* Naive Bayes Classification
* Support Vector Machines
* Apply Natural Language Learning Techniques
* K Nearest Neighbor Algorithm

These skills are practiced through the following tasks. The directions are generalized for the purpose of this report:

* Using any dataset from the given sheet
  + plot how many items are in each category
  + create a Naive Bayes Classification and evaluate
* Create a SVM classification
  + Apply SVC with various different options
  + Report the accuracy of the various models
* Apply natural language learning techniques to a txt file
  + Lemmatization
  + bigrams
    - calculate the frequency of each bigram
    - find top 5 most frequent bigrams
    - output all sentences that contain one of the top 5 bigrams
* Explore the KNN algorithm and show how accuracy changes based off of K
  + provide explanation for the changes

**FEATURES**

Task 1:

***Perform Naive Bayes classification on a dataset. Graph the frequency of the various categories.***

The script loads the digits\_dataset. It uses that data set to create a bar graph showing how many images represent each digit (0-9). It then performs a Multinomial Naive Bayes classification on the dataset. Out of the three most commonly used Naive Bayes classifications, (Guassian, Multinomial, and Binomial), Multinomial had the consistently highest accuracy than the other two models.

Task 2:

The program loads the iris data set and splits it into data and target. It then creates a training set and test set. Multiple variables are then used for the gamma and C parameters and the results are evaluated

Task 3:

***Perform various natural language learning tasks on a .txt file***

The program loads in a .txt file from the same directory as the .py files. It tokenizes the string. The script outputs the results from a Lemmatizer and saves it to a separate txt file. The bulk of the program is to find and handle the bigrams in the file. To make the bigrams more interesting, the code strips out as much of the common punctuation as possible. The bigrams are output to a .txt file. They are also counted, and the most frequent are stored in a separate container. The original text is read back in (so it includes the original punctuation). The text is then crawled to find the bigrams. If a sentence (split by ‘.’) contains one of the most frequent bigrams it is output to another .txt file.

Task 4:

The program loads the iris data set and splits it into data and target. It then creates a training set and test set. The data is then used in the K Neighbors Classifier model. The program then classifies 3 random iris values I entered and then also returns the accuracy score for the data model.

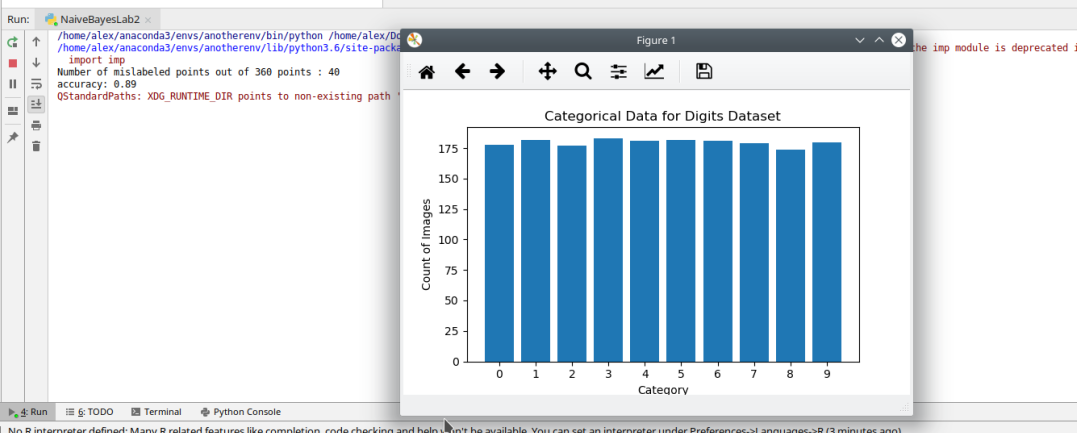
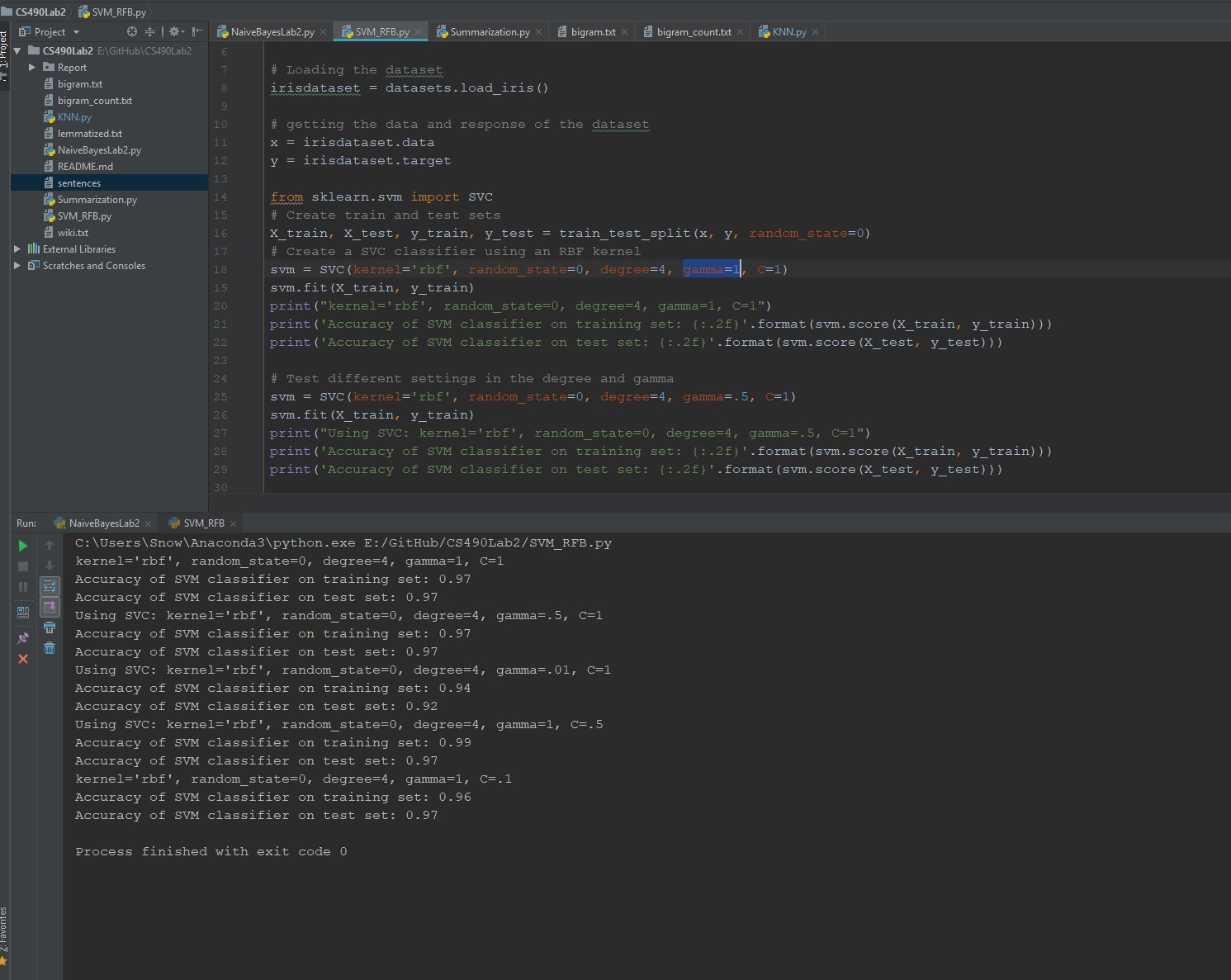
**CONFIGURATION**

All of the code in this lab was written and built using PYCHARM IDE, in an Anaconda3 environment and using Python 3.6.

**INPUT/OUTPUT SCREENSHOTS**

Task 1:

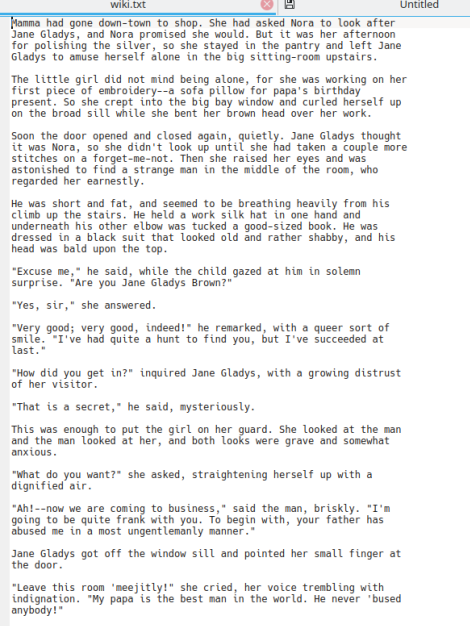
There is no user input for this task. It reads in data from a dataset in the sklearn library. The below screenshots show the outputted graph as well as the analysis of the Multinomial Naive Bayes Classification.

Task 2: Straight forward just run the program

Task 3:

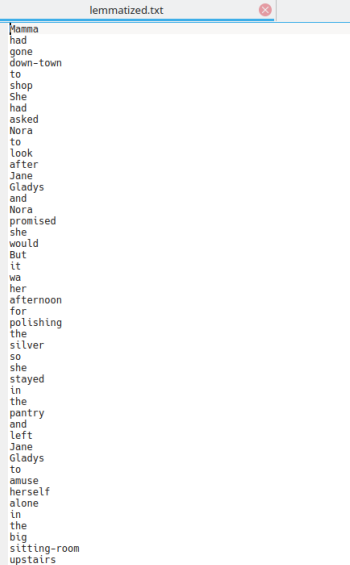
This task reads in wiki.txt file included with the source code. It contains a public domain children’s story. For a full look at the input please see our github repository. For space considerations, only clips of the input and output data of this task will be referenced here.

**Input:**

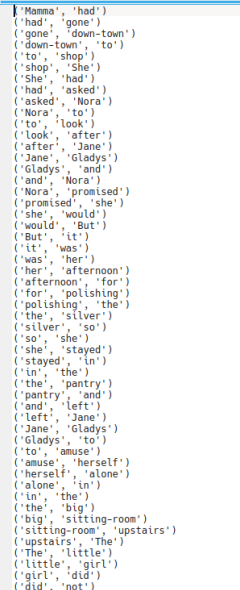


**Outputs:**

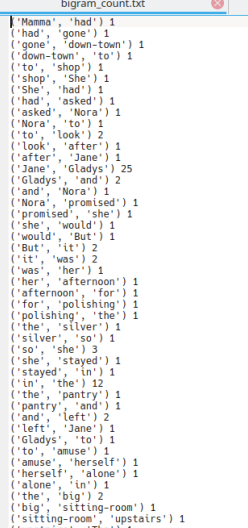
Lemmatizer Results:



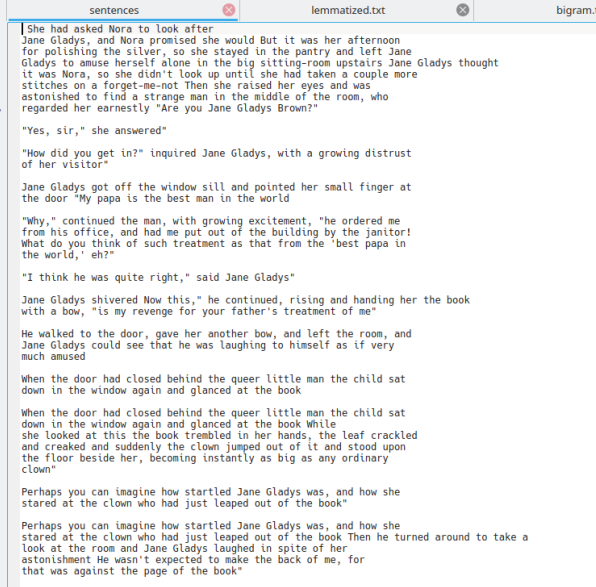
Bigrams (uncounted)



Bigram (counted – most frequent at the bottom, but not shown in this screenshot)

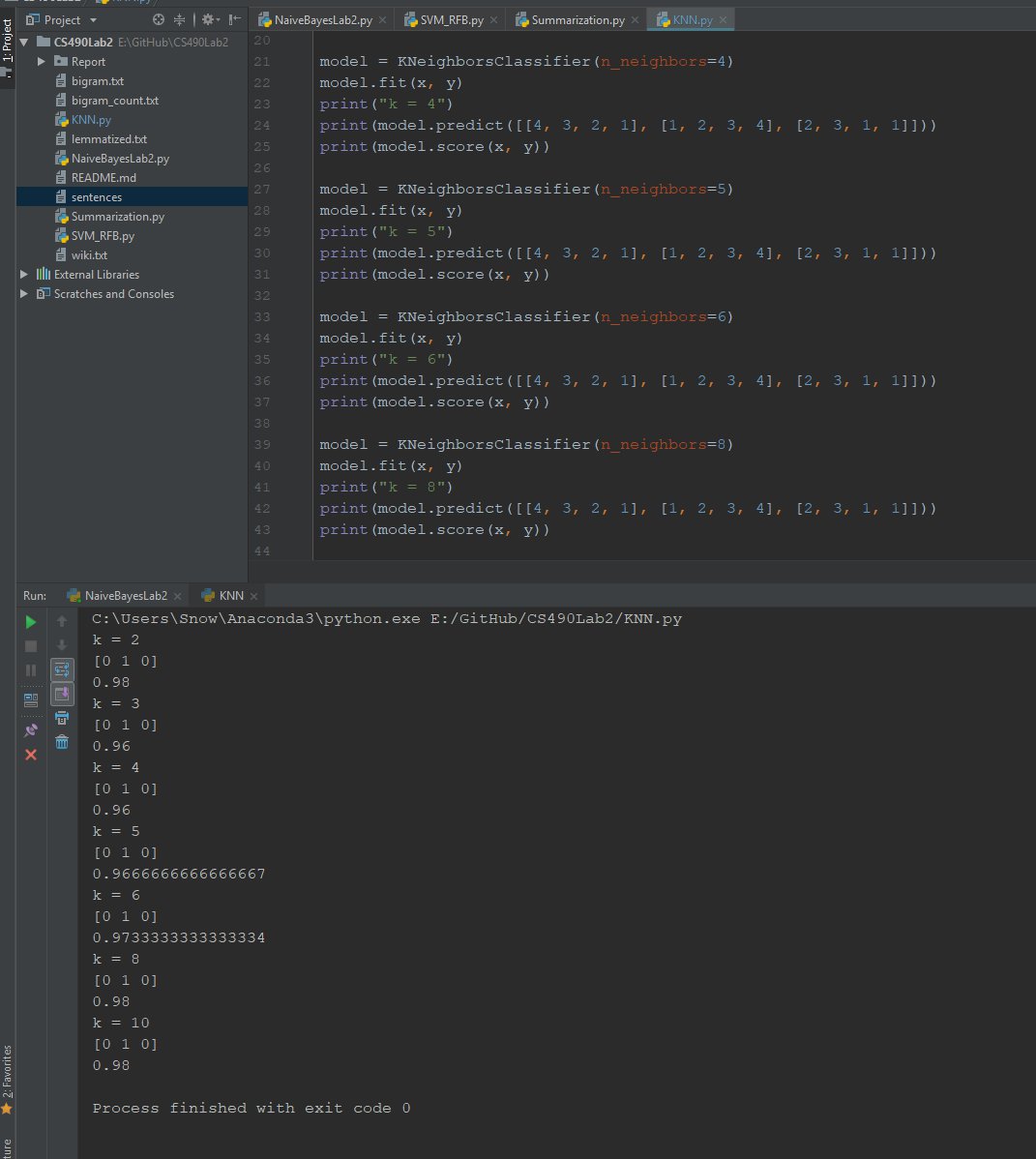


Sentences containing the five most frequent bigrams:



Task 4:

Just run the program



**IMPLEMENTATION & FULL SOURCE CODE**

Task 1:

To implement a Naive Bayes classification of a dataset, most of the legwork is done by the SKLEARN library. To count the number of times an image fit into one of the 10 categories, a simple counting function using a dictionary was implemented. Matplotlib was used to produce the bar graph of the category frequencies.

**from** sklearn **import** datasets

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.naive\_bayes **import** MultinomialNB

**from** sklearn.metrics **import** accuracy\_score

**import** matplotlib.pyplot **as** pp

*# Summary: Creates a bar graph based of the digit images and which digit they correspond to*

*# param1: <name>countedImages</name> expecting a dictionary / function that returns a dictionary. Uses keys and*

*# values to create bar graph of categories*

**def** createBarGraph(countedImages):

pp.bar(range(len(countedImages)), list(countedImages.values()), align= **'center'**)

pp.xticks(range(len(countedImages)), list(countedImages.keys()))

pp.xlabel(**'Category'**)

pp.title(**'Categorical Data for Digits Dataset'**)

pp.ylabel(**'Count of Images'**)

pp.show()

*# Summary: Counts the number of times a digit is represented in the dataset*

*# param1: <name>y</name> expecting an array of digits represented in the dataset*

*# returns: dictionary of the digits and the number of times they are present in the dataset*

**def** countCategories(y):

countOfImages = {}

**for** digit **in** y:

**if** digit **not in** countOfImages:

countOfImages[digit] = 0

countOfImages[digit] = countOfImages[digit] + 1

**return** countOfImages

*# Loads dataset*

digits\_dataset = datasets.load\_digits()

x = digits\_dataset.data

y = digits\_dataset.target

*# Split data into test and training set*

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2)

*# MultinomialNB*

*# used Multinomial over Gaussian because it has better / more consistent accuracy*

mnb = MultinomialNB()

*# train*

mnb.fit(x\_train, y\_train)

*# predict test set*

predict\_values = mnb.predict(x\_test)

*# find accuracy of model*

print(**"Number of mislabeled points out of "** + str(x\_test.shape[0]) + **" points : "** + str((y\_test != predict\_values).sum()))

accuracy = accuracy\_score(y\_test, predict\_values, normalize=**True**)

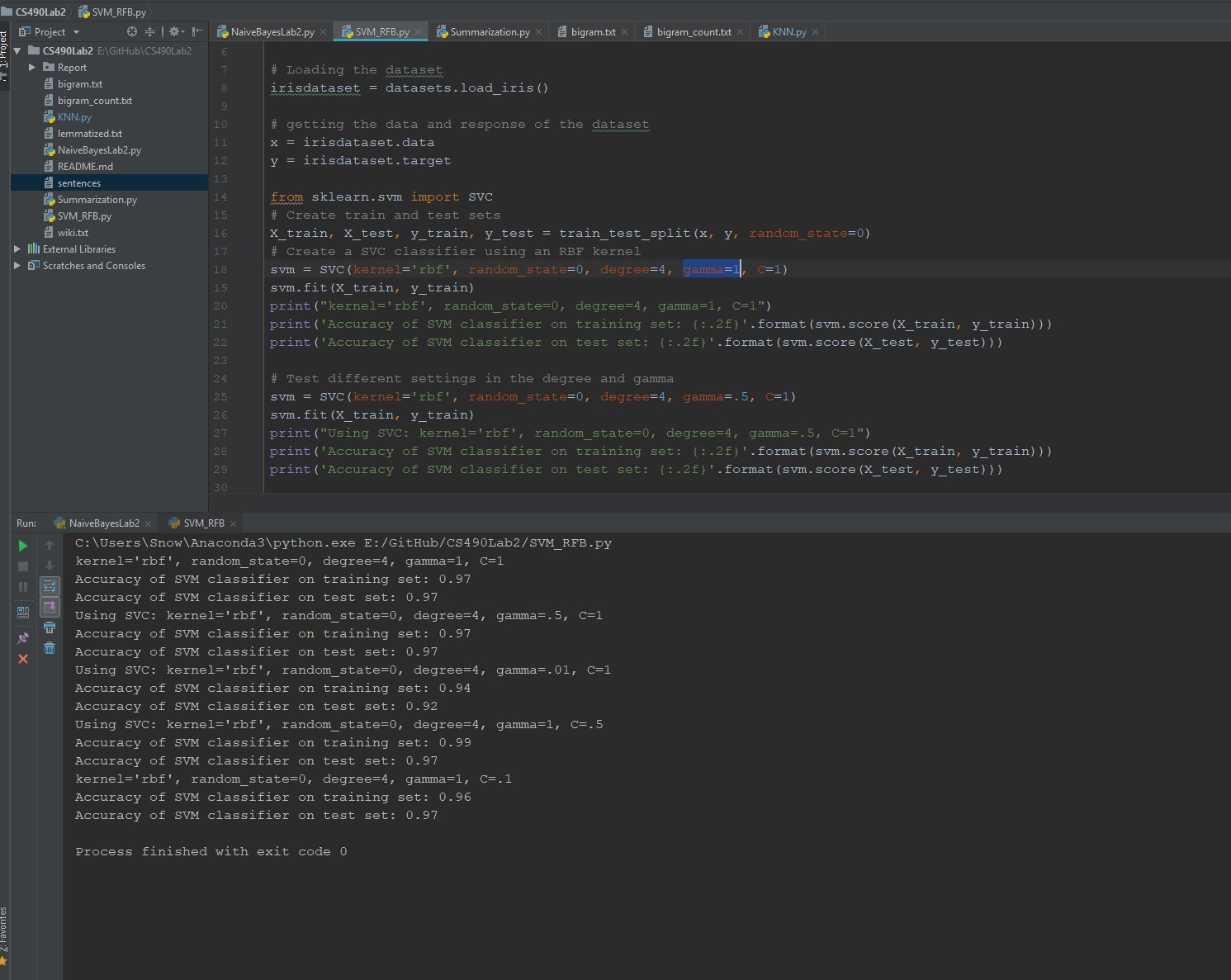
rounded = round(accuracy, 2)

print(**"accuracy: "** + str(rounded))

createBarGraph(countCategories(y))

Task 2

Implementation is straight forward. The dataset is split into x , y : the data and targets. The data and target sets are then split into train and test sets. From there scikitlearn does the hard work. By importing SVC only the parameters of the function need to be changed



Task 3:  
The implementation for the natural language learning is primarily done by use of built in functions in the nltk library. Reading in the text files and outputting to the text files are done by the built in python I/O functions. There are built in functions for Tokenizing the text and Lemmatizing the tokens. The implementation of the bigrams is a bit more complex. I used the bigram functionality in the nltk library. There are other libraries for using bigrams and scoring them in various ways. For the purpose of this lab, I found it much simpler to just use nltk.bigrams in combination with collections.Counter. I found the dedicated bigram library to be missing some valuable functionality I needed. I also found using that library made it more difficult to accomplish my goals for the lab. I found converting the bigrams to more usable data types to make searching the text, much easier.

**import** nltk

**from** nltk.stem **import** WordNetLemmatizer

**from** collections **import** Counter

nltk.download(**'wordnet'**)

nltk.download(**'punkt'**)

*# Summary: import txt file, remove punctuation*

*# Returns: check: string containing contents of txt file with out common punctuation*

**def** ImportFile():

file\_in = open(**"wiki.txt"**, **'r'**)

*# Remove as much punctuation as possible (makes better bigrams)*

punct = **".,;:''?``"**

replace = **" "**

trans = str.maketrans(punct, replace)

text = file\_in.read()

check = text.translate(trans)

*# Return text to be tokenized*

**return** check

*# Summary: tokenize the text*

*# Param: file\_text - string to be tokenized*

*# Returns: word\_tokens*

**def** CreateTokens(file\_text):

word\_Tokens = nltk.word\_tokenize(file\_text)

**return** word\_Tokens

*# Summary: lemmentize the word tokens, output the results to a file*

*# Param: words - tokens*

**def** Lemmentize\_Words(words):

lemmentizer = WordNetLemmatizer()

output\_file = open(**"lemmatized.txt"**, **'w'**)

**for** token **in** words:

output\_file.write(lemmentizer.lemmatize(token) + **'\n'**)

*# Summary: creates bigrams- counts them - finds 5 most common bigrams - find sentences containing most common*

*# bigrams and output them to a file.*

*# Param: words - tokenized text*

**def** Bigrams(words):

*# File Management*

output\_file = open(**"bigram.txt"**, **'w'**)

output\_file2 = open(**"bigram\_count.txt"**, **'w'**)

output\_file3 = open(**"sentences"**, **'w'**)

input\_file = open(**"wiki.txt"**, **'r'**)

file\_in = input\_file.read()

*# Create bigrams*

bigram\_tokens = nltk.bigrams(words)

*# convert to tuple for easier use*

list\_bigram = tuple(bigram\_tokens)

*# Output full list of bigrams to file*

**for** bigram **in** list\_bigram:

output\_file.write(str(bigram) + **'\n'**)

*# Count the frequency of each bigrm*

bigram\_counter = Counter(list\_bigram)

*# Output the counts to a file*

**for** item ,count **in** bigram\_counter.items():

output\_file2.write(str(item) + **' '** + str(count) + **'\n'**)

output\_file2.write(**"\n\n\n"**)

output\_file2.write(**"Most frequent bigrams:"** + **'\n'**)

*# Output the top 5 most frequent bigrams & output*

freq\_bigrams = tuple(bigram\_counter.most\_common(5))

**for** list\_item **in** freq\_bigrams:

output\_file2.write(str(list\_item) + **'\n'**)

*# Return only the bigram (strip the count)*

search\_bigrams = []

**for** f, f2 **in** freq\_bigrams:

search\_bigrams.append(tuple(f))

*# Search the original text for the bigrams, make list of the sentences*

sentences = []

check = 0

**for** sent **in** file\_in.split(**'.'**):

check += 1

**for** item\_a, item\_b **in** search\_bigrams:

test = str(item\_a) + **' '** + str(item\_b)

**if** test **in** sent:

sentences.append(sent)

*# Print the sentences to a file*

**for** s **in** sentences:

output\_file3.write(str(s))

text\_file = ImportFile()

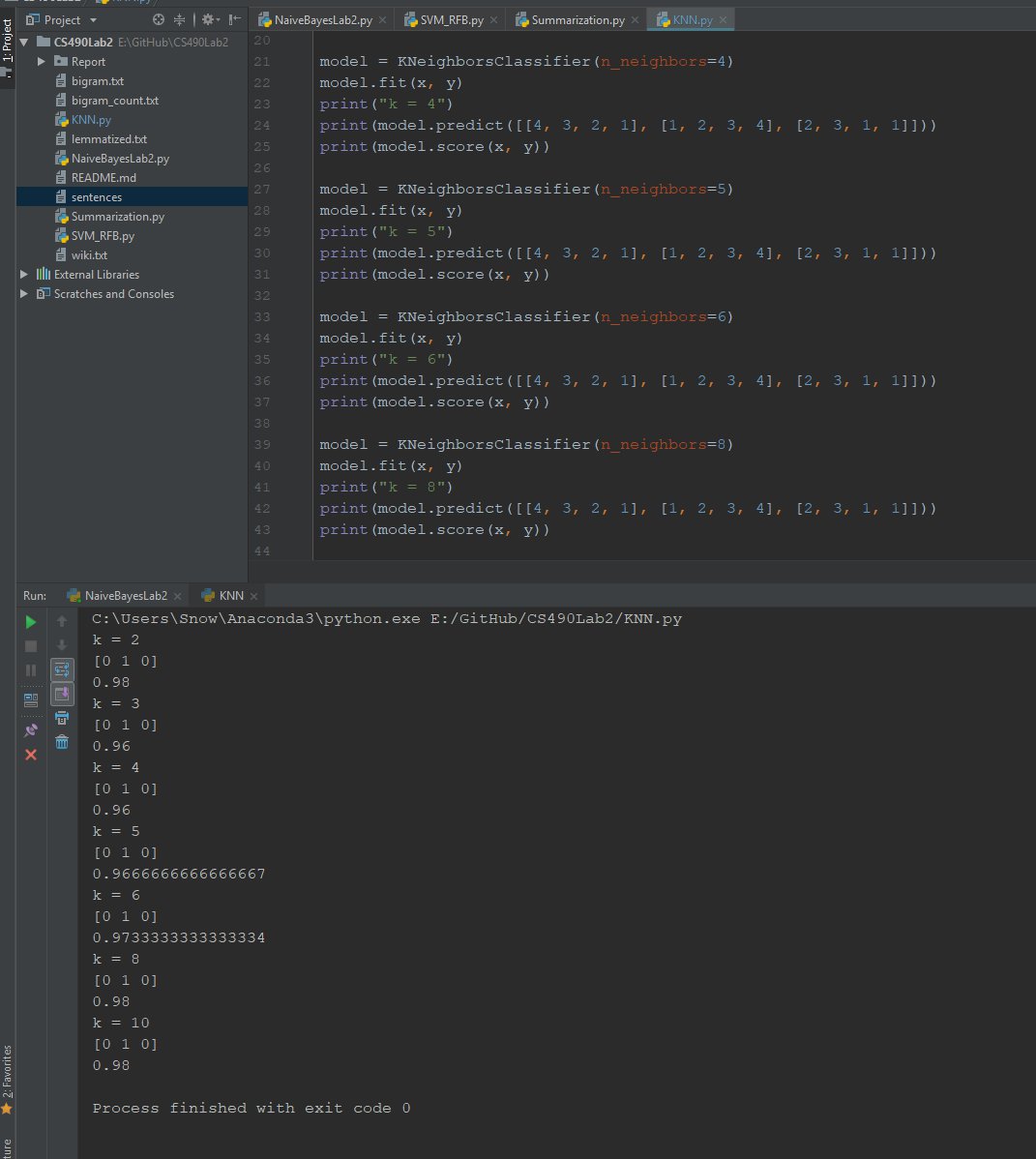
tokens = CreateTokens(text\_file)

Lemmentize\_Words(tokens)

Bigrams(tokens)

Task 4:

Implementation is straight forward. The dataset is split into x , y : the data and targets. The data and target sets are then split into train and test sets. From there scikitlearn does the hard work. By importing KneighborClassifier only the parameters of the function need to be changed



**REFERENCES**

<https://docs.python.org/3/tutorial/inputoutput.html>

<https://docs.python.org/3.1/library/collections.html>

https://stackoverflow.com/