Collins Kariuki.

Assignment 09.

ANAGRAMS.

- Map.

Input: key = IntWriteable, value = Text
Output: key = Text, value = Text

1. Divide the text input into individual words.
2. You have the option to store these individual words in a list.
3. Take each word from the list and arrange its characters in alphabetical order.

- Reduce

Input: key = Text, value = Text
Output: key = Text, value = Text

1. Go through the sorted words stored in a list.
2. Examine whether the characters of each sorted word match any words in the input set, such as checking if "act" matches "cat."
3. In case of a match, designate the sorted word's characters as the key and associate it with the corresponding original words that match in the input.

KNN (ALMOST).

- Map.

Input: key = IntIterable, value = Text
Output: key = IntIterable, value = Text

1. Go through the list of training examples and read each example using readExample.
2. Use getDistance to calculate the distance between the current example and test.
3. Set the value obtained from getDistance as the key and assign the label of the currently read example as the corresponding text value.

- Reduce.

Input: key = IntIterable, value = Text
Output: key = Text, value = IntWritable

1. We have a dictionary containing key-value pairs, where the key represents the distance, and the value is the label of the example.

2. Arrange the dictionary based on the keys, ensuring that the key:value associations are maintained during the sorting process.
3. Provide the resulting sorted key-value pair dictionary as the output.

NAÏVE BAYES (ALMOST).

- Map.

Input: key = Text, value = Text
Output: key = Text, value = IntWritable

1. Iterate through each line in the text to extract the label and example content, considering each line as we progress.
2. Within each example, traverse through the words, encompassing both the label and other words.
3. Create a key-value dictionary where the key is the label of the example appended to the current word being parsed, and the value is initially set to 0 <label+currentWord, 0>.

- Reduce.

Input: key = Text, value = IntWritable
Output: key = Text, value = IntWritable

1. Tally the occurrences of each word in every example, incrementing the value in the input key-value pair. If a word is found, transform <label + currentWord, 0> to <label + currentWord, 1>. The value represents the count of occurrences for that feature/label combination.

NAÏVE BAYES (EVEN CLOSER).

- Map.

Input: key = Text, value = Text
Output: key = Text, value = IntWritable

1. Traverse each line in the text, extracting the label and thus the corresponding example for each iteration.
2. Create a dictionary where the key is the label, and the value denotes the count of occurrences of that label in the text. Initialize the count to 0.

- Reduce.

Input: key = Text, value = IntWritable
Output: key = Text, value = IntWritable

1. Examine each example (line) and increase the input value by 1 whenever the label is encountered.
2. Provide the ultimate key-value pair, reflecting the count of occurrences for the labels.

FEATURE NORMALIZATION.

- Map 1.

Input: key = LongWritable, value = Text
Output: key = Text, value = IntWriteable

1. Iterate through each example to extract the features and their corresponding label.
2. To consolidate all feature values for a particular feature, consider using a key-value structure where the key represents the feature name, and the value is a list containing all feature values associated with that feature.
3. For each key-value pair, calculate the length of the value, representing the count of feature values, and store these lengths in a separate list. Note that it is certain that the length of this list matches the number of key-value pairs.

- Reduce 1.

Input: key = Text, value = IntWriteable.
Output: key = Text, value = IntWritable.

1. Iterate through the input set of key-value pairs.
2. While iterating, accumulate the sum of values within the lists associated with each key-value pair.
3. Simultaneously, access the list (by indexing into it using the iterator) storing the lengths of the lists (see step 3 of map 1).
4. Update the value in the original key-value pairs with the computed mean, calculated as the sum divided by the length (could be an instance variable). The resulting key-value pairs will take the form of <featureName: mean>.

- Map 2.

Input: key = Text, value = Text
Output: key = LongWriteable, value = Text

1. Loop through the input text and get the feature values and label.

- Reduce 2.

Input: key = LongWritable, value = Text
Output: same as input.

1. Utilize the previously calculated means.
2. Apply mean centering to each feature using the corresponding mean and the mean-centering equation.
3. Set the label obtained from map 2 as the key and assign the new mean-centered feature values as the value.