COMP9318 Project Report

- **Author : Yunqiu Xu**
- **ID : z5096489**

---

1. Introduction

- My model combines decision tree and KNN
- The dataset is splitted by number of vowels
- For num_vowels == 2, decision tree is implemented
- For num_vowels == 3, KNN is implemented
- For num_vowels == 4, KNN is implemented

2. Data Preprocessing

2.1 Preview the Raw Data

- Raw data consists of 50000 words, and the number of vowels ranges from 2 to 4. I split the raw data by their vowel number and shown some basic information in the table below:

Subset	Words	Max Word Length	Min Word Length	Target At 1	Target At 2	Target At 3	Target At 4
Vowel2	27619	11	2	24435	3184	-	-
Vowel3	16395	13	3	8938	6903	554	-
Vowel4	5986	14	5	1441	2135	2356	54
Unsplitted	50000	14	2	34814	12222	2910	54

- From the table we can see that the distribution of targets is not uniform that over 90% of the words have primary stress at Position 1 or Position 2, and Position 3

and Position 4 are unlikely to be predicted.

- By checking the words we can find the words are in different tenses (E.G. **PUTTING**, **SPOTTED**) and some words may be in plural form (E.G. **CAUSALITIES**) and the third person singular
- Some loan words are also in the dataset, whose pronounciation rules may be rather different from English. For instance, **BEIJING** from Chinese, **KIYOSHI** from Japanese and **GRZYWINSKI** from Russian

2.2 Transform into Normal Form

- I split the input data into word and phoneme list first:

```
>>> split_to_list('SOBRIQUET:S OW1 B R AH0 K EY2')
>>> ('SOBRIQUET', ['S', 'OW1', 'B', 'R', 'AH0', 'K', 'EY2'])
```

- Then by checking the position of vowels we can get phoneme list with stress tag removed and target:

```
>>> check(['S', 'OW1', 'B', 'R', 'AH0', 'K', 'EY2'])
>>> (['S', 'OW', 'B', 'R', 'AH', 'K', 'EY'], 1)
```

- Then some functions are are used to transform words in normal form:
  - singular_filter: remove "S/ES" from word
  - neutral_filter1: remove "ED/ING" and some neutral suffixes from word

```
>>> singular_filter('TRIGLYCERIDES',['T','R','AY', 'G', 'L', 'IH', 'S', 'ER', 'AY', 'D', 'Z'])
>>> ('TRIGLYCERIDE', ['T', 'R', 'AY', 'G', 'L', 'IH', 'S', 'ER', 'AY', 'D'])
>>>
>>> singular_filter('COUNTESSES', ['K', 'AW', 'N', 'T', 'AH', 'S', 'IH', 'Z'])
>>> ('COUNTESS', ['K', 'AW', 'N', 'T', 'AH', 'S'])
>>>
>>> singular_filter('YOURSELVES',['Y','UH', 'R', 'S', 'EH', 'L', 'V', 'Z'])
>>> ('YOURSELF', ['Y', 'UH', 'R', 'S', 'EH', 'L', 'F'])
>>>
>>> neutral_filter1('MUSCLING',['M', 'AH', 'S', 'AH', 'L', 'IH', 'NG'],
```

```
    vowel)
11. >>> ('MUSCL', ['M', 'AH', 'S', 'AH', 'L'])
```

- Words whose length is no more than 3 will not be transformed to avoid the abbrevations (E.G. **SOS**)
- Some special words will not be transformed as well(E.G. **BEIJING**)
- After this step we can get words in normal form and some words' target will exactly be 1 (E.G. **FLOCKING:F L AA1 K IH0 NG** $\Rightarrow$ **'FLOCK', ['F', 'L', 'AA', 'K']**)

2.3 Handle Compound Words

- For a lot of compound words, their suffix words can be removed to simplify the prediction. E.G. **BLUEBIRD:B L UW1 B ER2 D** can be refined as **BLUE:B L UW1**
- So a "compound_filter" is built to collect compound words and determine whether the suffix part should be removed.

```
1. >>> compound_filter('STEPCHILD', ['S', 'T', 'EH', 'P', 'CH', 'AY', 'L',
   'D'])
2. >>> ('STEPCHILD', ['S', 'T', 'EH', 'P')
```

3. Feature Extraction

- Then the dataset will be splitted as 3 subsets by NumVowels: Vowel2, Vowel3, Vowel4, those with no more than 1 vowel will be omitted for that the target will exactly be 1.

3.1 Features for Vowel2

- Since there are too many "similar" words with 2 vowels, it's not accurate to make prediction via KNN or LR, thus decision tree model will be implemented on this dataset, and the features are listed below (take **PERVIEW** for example):

Feature	**Example**
Vowel1	ER

Feature	Example
Vowel2	UW
V1Position	1
V2Position	4
V1Prefix	P
V1Suffix	V
V2Prefix	Y
V2Suffix	-

3.2 Features for Vowel3 and Vowel4

- For words with 3 or 4 vowels, we can try to simulate the process of human learning, that is, try to learn the rules of pronounciation and make prediction by finding those similar ones, thus KNN will be implemented on this dataset
- The features are listed below (take **ENSINGER** for example):

Feature	Explanation	Example
C123	Combination of all 3 syllables	EH N S IH N JH ER
C23	Combination of the 2nd and 3rd syllables	S IH N JH ER
C12	Combination of the 1st and 2nd syllables	EH N S IH N
C3	3rd syllable	JH ER
C2	2nd syllable	S IH N
C1	1st syllable	EH N
VowelPosition	Positions of vowels in binary form	[1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
ConsonantPosition	Positions of consonants in binary form	[0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Feature	Explanation	Example
Occurrence	Occurence of phoenmes in binary form	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, ...]
VowelsBin	The kinds of vowels	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...]
PrefixSuffixBin	The kinds of prefix and suffix for each vowel	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...]

- The features of Vowel4 are similar, with some changes(E.G. Combinations of syllables becomes [C1234,C234,C123,...])

3.2.1 Combination of Syllables

- I use the combination of syllables to refine the dataset:
  - For each test sample compute the combinations
  - Search words containing C123, then set this subset as training set
  - If this subset does not exist, try to find those containing C23
  - ...
  - If no syllable combination is finded, use the entire dataset
- In this way I can run model on those with similar syllables, which can improve the accuracy. The combinations is in descending order for that suffix syllables (E.G. **TION**) are more reliable than prefix ones(E.G. **RE**)

3.2.2 Position of Phonemes

- Both the positions of vowels and consonants are computed in binary form
- As the length of words may be different, I set the maximum length as 15, if this position does not have phoneme, it will be set as 0 in both VowelPosition and ConsonantPosition

3.2.3 Occurrence of Phonemes

- A binary list is set to represent the kinds of phonemes appear in current word
- Count list will not be used for that the distance will be overestimazed if there are a lot of same phonemes

3.2.4 VowelsBin and PrefixSuffixBin

- We need to know the kind of vowel at each position, and its predecessor and

successor
- A binary list(length = 15) is set to represent the vowel at current position
- A binary list(length = 40) is set to represent the prefix/suffix at current position. All vowels and consonants are included, an additional position is set to represent the condition of no prefix or suffix,

3. Experiment

3.1 Training

- In training process, each row_data will be preprocessed and then assigned to one of 3 subsets.
- For Vowel2, decision tree is built and trained:

```
tree2 = DecisionTreeClassifier(criterion = 'entropy')
tree2.fit(X2,y2)
```

- As KNN does not have explicit training process, we just build dataset Vowel3, Vowel4
- Then tree model and dataset Vowel3, Vowel4 will be saved for testing

3.2 Testing

- In testing process, each row_data will be preprocessed first. If there is no more than 1 vowel in the phoneme list, it will be predicted as 1 directly. Some special cases will also be considered, that is, if there is only one vowel but the prefix is in `^NON|^MC|^SUB|^RE|^TRAN|^PRE|^FORE|^WITH|^[AEIOU]|^DIS|^MIS|^CON|^CUR|^SUR`, it should be predicted as 2.
- For words with more than 1 vowels, some special cases will also be considered before using LR model. If some special suffixes are matehed, the prediction will be made without using model
- If num_vowel == 2, decision tree model will be loaded and make prediction
- If num_vowel == 3:

- The refined dataset will be built first by matching similar combination of syllables
- Then distance-weighted KNN classifier will be built to make prediction
- n_neighbours is set as 13, if there are less than 13 samples in refined dataset, n_neighbours will be set as the number of samples

```
if refined_df.shape[0] < 13:
    neigh = KNeighborsClassifier(n_neighbors = train_X.shape[0], weights = 'distance')
else:
    neigh = KNeighborsClassifier(n_neighbors = 13, weights = 'distance')
```

- If num_vowel == 4:
  - The refined dataset will be built first by matching similar combination of syllables
  - Then distance-weighted KNN classifier will be built to make prediction
  - n_neighbours is set as 15, if there are less than 15 samples in refined dataset, n_neighbours will be set as the number of samples

```
if refined_df.shape[0] < 15:
    neigh = KNeighborsClassifier(n_neighbors = train_X.shape[0], weights = 'distance')
else:
    neigh = KNeighborsClassifier(n_neighbors = 15, weights = 'distance')
```

3.3 Experiment

- The experiment is performed via 10-fold CV, for each subset the training accuracy, testing accuracy and f1-score are calculated. Finally the average result will be shown
- Some other models I built before will also be shown to make comparation, the parameters of each model have been optimized

4. Result and Discussion

Model Index	Description	Training Accuracy	Testing Accuracy	Precision	Recall	F1
1	Unsplitted, Tree	0.9988	0.8863	0.7099	0.6567	0.6761
2	Splitted, Tree	0.9921	0.8975	0.6752	0.6830	0.6787
3	Unsplitted, LR	0.9719	0.9178	0.8030	0.7341	0.7580
4	Splitted, LR	0.9797	0.9078	0.8654	0.8376	0.8447
5	Splitted , KNN model	0.9945	0.9157	0.7666	0.7047	0.7227
6	Splitted , LR for vowel2 , KNN for vowel3 & vowel4	0.9938	0.9069	0.8488	0.8326	0.8179
7	**(FINAL)** Splitted , Tree for vowel2 and KNN for vowel3 & vowel4	0.9964	0.9033	0.8561	0.8564	0.8558

- From the result table we can see that splitted dataset with the combination of decision tree and KNN achieves best performance.
- Tree model is not stable that by using differnt datasets there can be large fluctuations of performance, and the average F1 can be low. However it performs best in the subset Vowel2, a possible reason is that there are too many "similar" words in this dataset and it's not easy to find the nearest neighbour accurately.
- By using logistic regression the average precision will be improved, however the recall is still low. LR model is prone to predict a sample as 1, thus the precision is high for that most samples' targets are 1. However the recall of 3 and 4 plays a more important role in improve the f1 score.
- In order to improve the recall, some pre-defined rules are used before adapting the model, and KNN is used for Vowel3 and Vowel4 to find those similar ones. The result is that the recall as well as F1 is improved , with acceptable decrease of precision.

5. Conclusion

- I choose the 7th model as final model: split the dataset by number of vowels, then decision tree for Vowel2 , and KNN for Vowel3 / Vowel4
- However the performance of model still need to be improved, from the result we can find that all models are suffering from overfitting. Some optimizations have been tried, but the testing accuracy did not improve.
- For later research, some other classifiers, such as SVM and neural network can be tested, and ensemble learning methods such as random forest and bagging can be implemented to find suitable parameters.

6. References

[1] Baptista B O. Strategies for the prediction of English word stress[J]. IRAL-International Review of Applied Linguistics in Language Teaching, 1989, 27(1): 1-14.
[2] Gillis S, Daelemans W, Durieux G. Lazy Learning': A Comparison of Natural AND MACHINE LEARNING OF Word STRESS[J]. 2000.
[3] James S. Learning English Stress Rules-Using a machine learning approach[J]. 1999.
[4] Kim Y J, Beutnagel M C. Automatic assessment of american English lexical stress using machine learning algorithms[C]//SLaTE. 2011: 93-96.
[5] Ross J R. A reanalysis of English word stress (Part I)[J]. Contributions to Generative Phonology, 1972: 229-323.
[6] Selkirk E O. The role of prosodic categories in English word stress[J]. Linguistic inquiry, 1980, 11(3): 563-605.