Rasik Kane **Text Analytics COMP47600** **08/11/2020**
19200172 **Practical 6**

Question 1 Check out the provided program on k-NN, applied to the famous Iris Data set Two parameters are interesting: (i) the split which is the size of the training subset v test subset (split = .67) means roughly 2/3rds training, 1/3rd testing, (ii) k which is the size of the collection of nearest neighbors used for the prediction. Note, the Accuracy of model is determined for test sets; it measures how well it classifications work for this unseen instance, that have been separated from the training set.

k-NN algorithm
k-NN algorithm is supervised classification algorithm working on principal of 'Vote Majority'. In traditional sense, implementation of k-NN in provided code does not 'fit' on training data to learn a model. Instead, while classifying a new test sample S, Euclidian distance of S from each of Training sample is calculated. And after sorting the distances, class labels of 'k' nearest training samples from S are returned. Majority vote determines the class of test sample. For example : if k = 3 AND nearest training set samples from test sample S are : [(d= 0.2, class = 1), (d= 0.25, class = 0), (d= 0.4, class = 1)] → Then sample S is classified as class 1 object.

1(a)(i) Systematically vary the size of the split ; exploring five other values for it >0.0 and <0.9 (to be chosen by you)

Train test split is performed on data with ratio varying successively and count of neighbors, k is equal to 3. As observed, when split ratio[i.e. training samples / total samples] increases, i.e. size training set increases; then testing accuracy initially increases. But after split ratio of 0.6, testing accuracy decreases. This is evidence that model is overfitting.

Split ratio	Accuracy
0.3	92.7835
0.4	95.5056
0.6	96.8254
0.8	96.15
0.9	95.45

1(a) (ii) Systematically vary k on 10 selected values between 1 and 50.

Train test split is performed on data with split ratio of 0.7s and number of neighbors, k is varied.
Since provided dataset has only 150 samples, we can choose split ratio only greater than or equal to ratio 1/3 for comparative analysis. Otherwise, size of training set would be less than 50 and we cannot check Euclidean distance of test sample for k=50

k-neighbours	Accuracy
3	92.1052
5	95.12195
8	96.0784
10	97.6744
15	97.6584
18	96.3488
20	94.7368
30	94.1176
40	95.4
50	92.85

1(b) In two separate graphs plot the accuracy scores you get for these parameter changes. For each of graphs , discuss the results found; explain why the graph goes up/goes down/is unchanging when the parameter is varied.

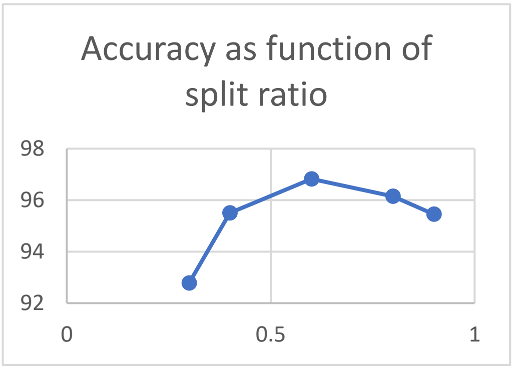


Figure 2 Effect of Split ratio on accuracy

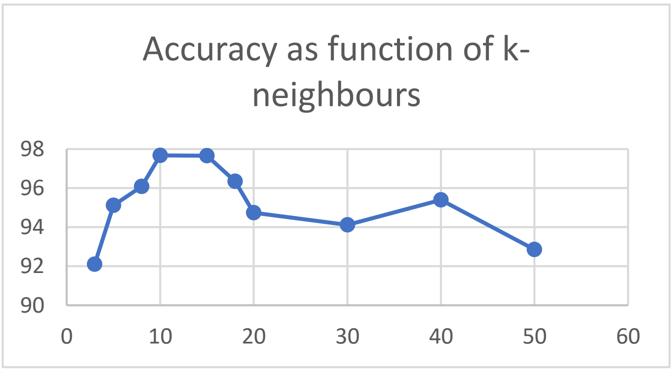
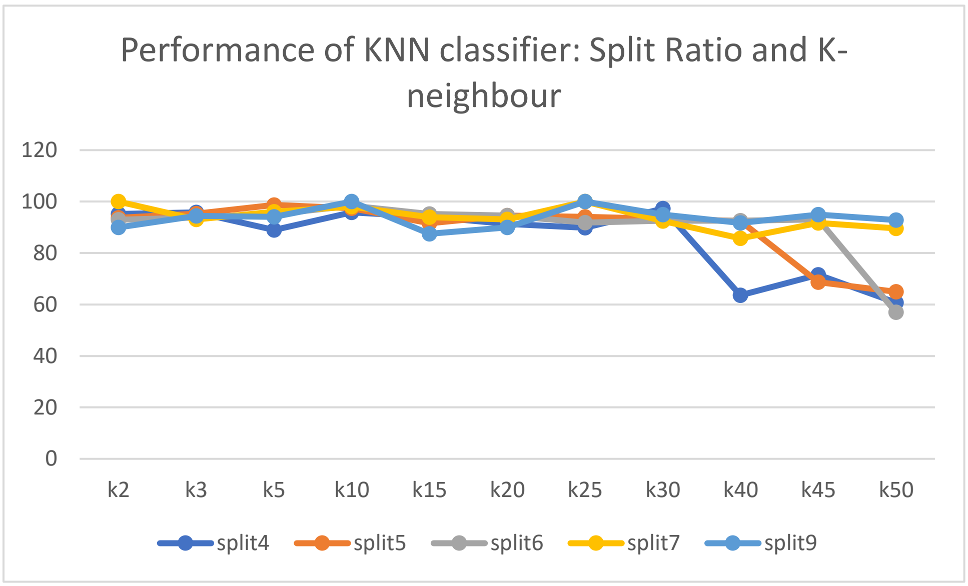


Figure 1 Effect of k-neighbors on accuracy

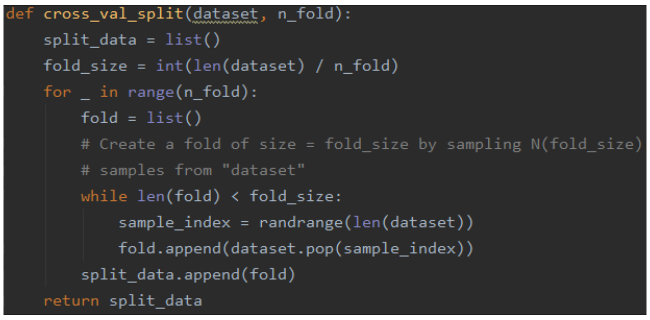
- When k is maintained constant at 3 and split ratio is varied over range 0.1 to 0.9, initial increase in accuracy is seen till split ratio of 0.6. But increase is stalled after split ratio indicating that split ratio would not have direct proportional influence. Instead, it would overfit model.
- Similarly, for split ratio 0.7, when value of k-neighbor exceeds 15, testing accuracy starts decreasing, which is indicator of underfitting.
- Overfitting is strong likeliness for taken corpus of Iris dataset, which contains only 150 samples. Hence, data is biased towards majority class in Training data.
- To verify this phenomenon, I calculated accuracies of all k-values for various split ratios above 0.34. In the graph, value k3 on X axis indicates 3-neighbors and split4 indicates 40:60 split ratio for training: testing data.



- I observed that, in general, models are underfitting for combination [low split ratio and higher values of k], which is consistent with definition of underfitting.
- Higher split ratio provide better acumen of data. This is translation of fact that dataset is small. So, predictions involving higher value of neighbors is effective only when enough training data is available.

Concept of Cross validation

In simple train-test split, classifier is trained on training data and tested on test data. So, model completely relies on training data and information content of test data is ignored. To train model using entire data, k-fold cross validation technique is used. In the process, dataset of corpus size C is divided into k sets of n random samples each. And to judge the performance of model, model is iteratively trained on (k-1) sets and tested on remainder set. This way, model is evaluated on entire data and we can ensure that it is not overfitting for certain distributions of training data. Finally, metrices are aggregated to comment on model performance. **To understand this concept, I ran cross validation on data.**



```
def cross_val_split(dataset, n_fold):
    split_data = list()
    fold_size = int(len(dataset) / n_fold)
    for _ in range(n_fold):
        fold = list()
        # Create a fold of size = fold_size by sampling N(fold_size)
        # samples from "dataset"
        while len(fold) < fold_size:
            sample_index = randrange(len(dataset))
            fold.append(dataset.pop(sample_index))
        split_data.append(fold)
    return split_data
```

- From Figure 3, we know that split ratio of 0.7 with k-neighbors between 5-10 best approximate iris dataset. Hence, I set k = 7 and n_fold = 4 → that would give split ratio of ~0.75
- Metric of classification returned after evaluating each fold are → 91.89, 99.01, 97.29, 94.59]
- This indicates that testing accuracies vary within span of 10%. This is indicator of overfitting. Probable solutions include data augmentation using oversampling or collecting more valid data.

Question 2 Have a look at the nltk Bayes Classifier that does the prediction of male/female names based on the last letter in the name. Think of a new feature that you could extract from the data-set; define a method for it (modifying gender features). Discuss the results of this change, what new outputs occur in the classification and what things remain the same? [Hint: there is a built-in method showing_most_informative_features which may be of help; read about it in nltk].

Problem
Problem posed is predicting gender of a person correctly when her/his name is supplied. List of names of male [2943 samples] and female [5001 samples] are parsed and set of (name, gender_label) tuples is made. Before applying classifier, this list of tuples is shuffled and divided into 2 groups : training and testing. 1st 500 tuples of shuffled list are held up for testing and rest of 7444 samples are used for training classifier.

Approach
Training the classifier on list of entire corpora of names would make it a mare memorizer. But names have patterns. E.g. Phonetic endings of male names are most often (Julius - ə). Whereas, for female names, vowel endings are seen frequently (Julia - ɑː). Hence, naively, name ending in 'a' such as 'Ophelia' is more likely to be of same class as Julia. Such patterns or **features** are captured using naïve bayes classifier of python's nltk package.

Naive Bays classifier
Bays theorem is basis of probability based supervised classification. In case of given data, It works on thought that if probabilities of class labels[G - gender], feature sets for names[$f_{1,2,..n}$: {'last_letter', 'is_last_letter_vowel', ...}] and conditional probability of feature set given class label is known – then we can predict a class label given feature set of new data. This aposteriori value [refer formula] is calculated for each class and class label that maximizes P(G | n) is assigned to the data.

$$P(G \mid f) = \frac{P(G) * P(f \mid G)}{P(f)}$$

Naive Bays classifier mitigates a potential challenge with Bays classifier that is – predicating conditional probability of all feature vector tuples given a label. Instead, Naïve Bayes Classifier considers all features conditionally independent. Which means –

- Suppose for a set of names [offilia, Julia, Aradhana], we find 2 features – ['last_letter', 'is_last_letter_a_vowel']
- We know that both these features must have correlation since 'aeiou' are captured as vowels as well as last letters.
- But for simplicity of calculations, Naïve Bayes classifiers ignore it and treat features separately[refer figure]

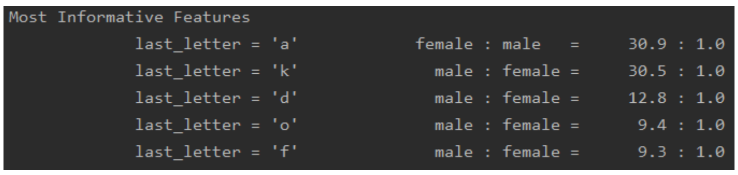
$$P([f_1, f_1, f_1, f_1, .. f_1] \mid G) = \prod_i P(f_i|G)$$

- In case of nltk package, denominator P(f) is not explicitly calculated. But instead – numerator is simply normalized. Hence, apriori for each class "label" given set of labels "l" and functions "f1..fn" is given by

$$P(label \mid feature) = \frac{P(label) * P(f1 \mid label) * \ldots . P(fn \mid label)}{sum(l) * [P(l) * P(f1 \mid l) \ldots . * P(fn \mid l)]}$$

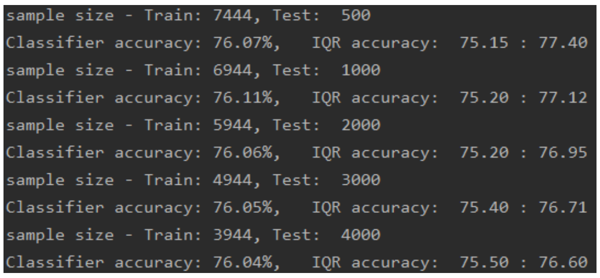
Present Implementation
Current implementation derives a single feature 'last_letter' from data. Hence, last letters of all names in training data are used as input features and mapped against class labels to train a naïve bayes classifier.

Observations
- 26 distinct cases [Counter({'last_letter': 26})] are found for feature 'last_letter'. [Roman letters a-z]
- Most informative last letters [which lead to a classification boundary] are [a, k, d, o, f]. As assumed, names ending with 'a' have high chance of classification as female name.



```
Most Informative Features
             last_letter = 'a'            female : male   =     30.9 : 1.0
             last_letter = 'k'              male : female =     30.5 : 1.0
             last_letter = 'd'              male : female =     12.8 : 1.0
             last_letter = 'o'              male : female =      9.4 : 1.0
             last_letter = 'f'              male : female =      9.3 : 1.0
```

- Accuracy varies each time program runs, hence I averaged accuracies over 100 iterations and calculated mean accuracy . This single feature approach gives **prediction accuracy of ~75-76% with default split ratio = 0.93** (test – 500): (train-7444)
- **When I varied split ratio; No significant change in mean accuracy was observed.**



```
sample size - Train: 7444, Test:  500
Classifier accuracy: 76.07%,   IQR accuracy:  75.15 : 77.40
sample size - Train: 6944, Test:  1000
Classifier accuracy: 76.11%,   IQR accuracy:  75.20 : 77.12
sample size - Train: 5944, Test:  2000
Classifier accuracy: 76.06%,   IQR accuracy:  75.20 : 76.95
sample size - Train: 4944, Test:  3000
Classifier accuracy: 76.05%,   IQR accuracy:  75.40 : 76.71
sample size - Train: 3944, Test:  4000
Classifier accuracy: 76.04%,   IQR accuracy:  75.50 : 76.60
```

Additional Features
I have added 2 new features to the feature selection function. Motive behind each feature is mentioned separately:

- Last_2_letters: Target male names like Nath**an** – Eth**an** – Ry**an**; Just**in** - Mart**in** etc.
- Last_3_letter: Target common female names like Ju**lia** – Ophe**lia,** Christ**ina** - Mart**ina** etc.

Observations

- Last_2_letters improves mean accuracy of classifier to ~78 % with default split ratio

My interpretation behind probable cause for increase in accuracy is : inherent bias associated last letter a, e and y with female names is not sufficient to classify male names like sacha, Hamja, Joe. 'last_2_letters' provides more level of details associated with combination of last to letters.

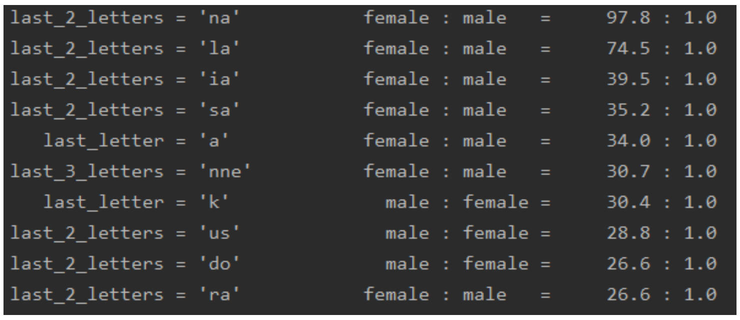
- Last_3_letters decreases mean accuracy of classifier back to ~75% with default split ratio

Decrease in accuracy probable result of underfitting. 'last_3_letters' would have 26^3 possible combinations. Such large training corpus is not available. We can observe Most Informative Features :

last_3_letters = 'nne'	female : male =	34.8 : 1.0	……… eg [Adrianne]
last_3_letters = 'ana'	female : male =	25.7 : 1.0	……… eg [Melina]
last_3_letters = 'tta'	female : male =	25.3 : 1.0	……… eg [Margretta]

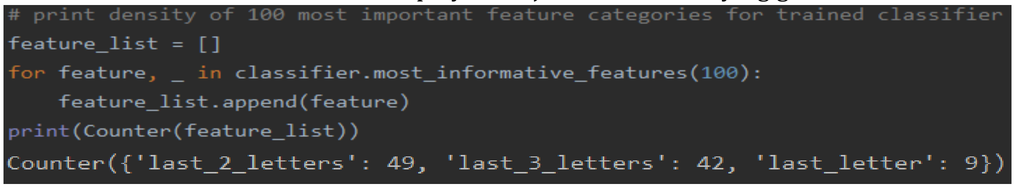
Importance of Features

To check effect of multiple features together to find optimal feature sets effecting classification. As new features are introduced, naturally vector combinations for those features are created by ML algorithm during fitting on training data. As last-letter had 26 possible vectors [a-z], similarly 'last_2_letters' would have 26^2 additional possible combinations.
But in naïve bayes, all features are considered conditionally independent; so finding effect of each feature on output is important for dimensionality reduction by omission during retraining. Classifier module of nltk package provides method show_most_informative_features(n). It returns top n important feature vectors which influence classifier. Top 10 features observed are shown in adjoined figure.



```
last_2_letters = 'na'          female : male   =     97.8 : 1.0
last_2_letters = 'la'          female : male   =     74.5 : 1.0
last_2_letters = 'ia'          female : male   =     39.5 : 1.0
last_2_letters = 'sa'          female : male   =     35.2 : 1.0
   last_letter = 'a'           female : male   =     34.0 : 1.0
last_3_letters = 'nne'         female : male   =     30.7 : 1.0
   last_letter = 'k'             male : female =     30.4 : 1.0
last_2_letters = 'us'            male : female =     28.8 : 1.0
last_2_letters = 'do'            male : female =     26.6 : 1.0
last_2_letters = 'ra'          female : male   =     26.6 : 1.0
```

Training multiple features endorsed accuracy to ~79% default split ratio. But cost of training [time] was much larger than individual features. When I analyzed frequencies in top 100 important features, I noticed that feature 'last_2_letters' and 'last_3_letters' played major role in classifying gender.



```
# print density of 100 most important feature categories for trained classifier
feature_list = []
for feature, _ in classifier.most_informative_features(100):
    feature_list.append(feature)
print(Counter(feature_list))
Counter({'last_2_letters': 49, 'last_3_letters': 42, 'last_letter': 9})
```

This solution can be improved by preprocessing input set. Joint names are not currently handled e.g. Hans-Peter, Jean-Paul, John-Patrick, Ann-Marie, Jo-Ann etc.