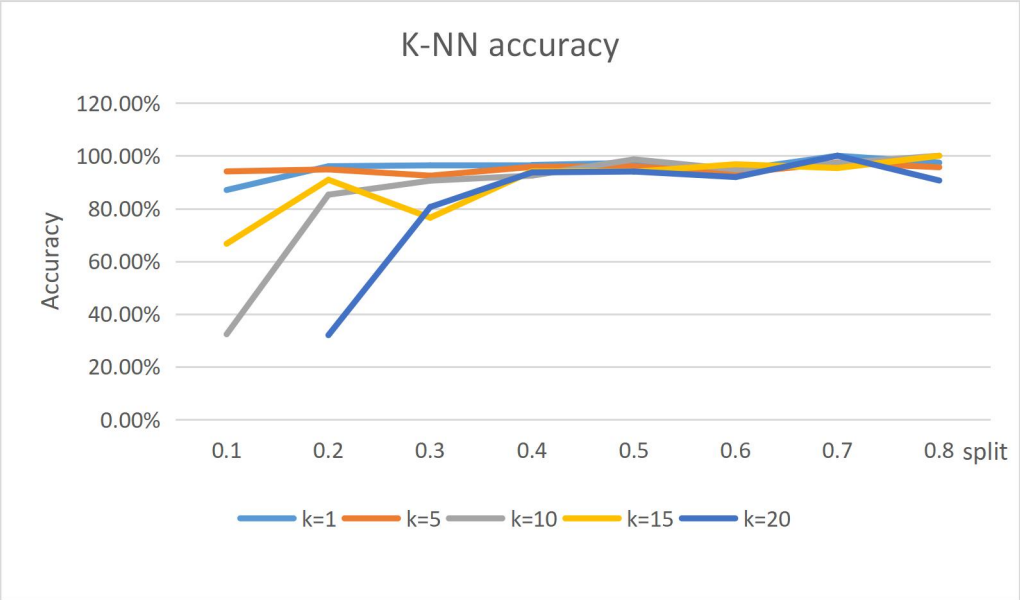
Q1: K-NN

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 nearest neighbours. Note, the Accuracy of model is determined for test sets; it measures how well it classifications work for the unseen set.

Taking ideas about cross-validation into account; you job is first to systematically vary the size of the split ; exploring a decent number of other values for it >0.0 and <0.9 (to be chosen by you). Also to systematically vary k on five selected values between 1 and 20.



In this question, I demonstrate the accuracy based on different K-NN and split. The 5 value of split should between 1 and 20, so I choose 1, 5, 10, 15 and 20. The split I chose is from 0.1 to 0.8, step by 0.1. This graph shows that when k =1 and k=5, the K-NN Classifiers have relatively higher accuracy , regardless of the value of split. When k=10, k=15 and k=20 , the graph shows that the higher the split value is, the higher the accuracy it will get. In my experiment, I found some accuracy such as when split=0.8 k=10 and k=15 the accuracy can be 100% which can be the best parameters in K-NN classifiers.

Then plot the accuracy in a graph for these parameter changes. Now, using this data set, describe an algorithm for doing a 5-fold cross validation on this data set. See can you implement it in Python.

In this question, the data set will be split into 5 folds. And we will run the K-NN method 5 times. In the first iteration, the first fold is used to test the model and the rest are used to train the model. In the second iteration,the second fold will be used as the testing set while the rest serve as the training set. This process is repeated until each fold of the 5 folds have been used as the testing set.Lastly, getting the accuracy scores of each test and selecting suitable parameters.

Q2: Bayes Classifiers

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 fn for it (modifying gender_features) and see what is learned from it in the classification.

Compare the accuracy of your feature versus that of the last_letter feature and discuss.

The accuracy when using last_letter as the selected features:

```
Mark is: male
Precilla is: female
Most Informative Features
             last_letter = 'a'            female : male   =     38.4 : 1.0
             last_letter = 'k'              male : female =     31.6 : 1.0
             last_letter = 'f'              male : female =     17.2 : 1.0
             last_letter = 'p'              male : female =     11.8 : 1.0
             last_letter = 'v'              male : female =     10.5 : 1.0
0.766
```

The accuracy when using first_letter as the selected features:

```
Mark is: female
Precilla is: female
Most Informative Features
            first_letter = 'W'              male : female =      4.5 : 1.0
            first_letter = 'U'              male : female =      2.9 : 1.0
            first_letter = 'Q'              male : female =      2.7 : 1.0
            first_letter = 'H'              male : female =      2.3 : 1.0
            first_letter = 'X'              male : female =      2.3 : 1.0
0.668
```

The accuracy when using second_letter as the selected features:

```
Mark is: female
Precilla is: female
Most Informative Features
           second_letter = 'k'              male : female =      6.6 : 1.0
           second_letter = 'z'              male : female =      6.5 : 1.0
           second_letter = 'j'              male : female =      5.1 : 1.0
           second_letter = 'c'              male : female =      3.2 : 1.0
           second_letter = 'p'              male : female =      2.7 : 1.0
0.628
```

The accuracy when using firsttwo _letters as the selected features:

```
Mark is: female
Precilla is: female
Most Informative Features
         firsttwo_letter = 'Hu'             male : female =     18.4 : 1.0
         firsttwo_letter = 'Ya'             male : female =     11.7 : 1.0
         firsttwo_letter = 'Wa'             male : female =     11.7 : 1.0
         firsttwo_letter = 'Tu'             male : female =     10.6 : 1.0
         firsttwo_letter = 'Fo'             male : female =      9.7 : 1.0
0.658
```

The accuracy when using lasttwo _letters as the selected features:

```
Mark is: male
Precilla is: female
Most Informative Features
          lasttwo_letter = 'na'           female : male   =     98.4 : 1.0
          lasttwo_letter = 'ia'           female : male   =     40.0 : 1.0
          lasttwo_letter = 'sa'           female : male   =     33.8 : 1.0
          lasttwo_letter = 'rt'             male : female =     33.7 : 1.0
          lasttwo_letter = 'ta'           female : male   =     31.4 : 1.0
0.754
```

After experiment , I got the outcomes as such:

When using the last letter as feature will have accuracy 0.766 and the prediction towards the example is correct.

When using the first letter as feature will have accuracy 0.688 and the prediction towards the example is half correct.

When using the second letter as feature will have accuracy 0.628 and the prediction towards the example is half correct.

When using the first two letters as feature will have accuracy 0.658 and the prediction towards the example is half correct.

When using the last two letters as feature will have accuracy 0.754 and the prediction towards the example is correct.

I also found that the last letter and the last two letters are the best feature to be used in this Bayes Classifiers. And when using two letter together, the Bayes Classifiers can produce the most informative features better.

Q3: SVMs

So, this is just a quick use of an SVM; to show you how easy it is. It involves learning digit recognition.

NB to install packages: sklearn, matplotlib, scipy; (problems around installing scipy in on mac with Pycharm; but works with "port install py34-scipy"

Now run three different training configurations and then test the outputs for 5-10 different digits; report results.

I choose this 3 configurations in order to control the variables. Based on this 3 classifier, test the outputs for 5-10 different digits;

```
classifier = svm.SVC(gamma=0.1, C=100)
```

```
classifier = svm.SVC(gamma=1000, C=100)
```

```
classifier = svm.SVC(gamma=0.1, C=0.1)
```

After experiment, I found that lower value of C can get more successful based on different digits. The prediction of the number will not be affected by the value of digits. I also found that The C parameter tells the SVM optimization how much I want to avoid misclassifying each training

example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable. A High value of Gamma leads to more accuracy but biased results and vice-versa. Similarly, a large value of Cost parameter (C) indicates poor accuracy but low bias and vice-versa.