Q1.) KNN

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. Then plot 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.

Solution -

KNN is a supervised learning algorithm. It is a non-parametric method which means that the algorithm doesn't make any assumption about the data. Here, we have labelled dataset consisting of training observations (x,y) and we would like to capture the relationship between x and y. More formally, the ultimate goal is to learn a function h : x → y so that given an unseen observation x, h(x) can confidently predict the corresponding output.

In the classification setting, the K-nearest neighbor algorithm essentially boils down to forming a majority vote between the K most similar instances to a given "unseen" observation. Similarity is defined according to a distance metric between two data points. A popular choice is the Euclidean distance.

KNN classifier performs the following two steps:

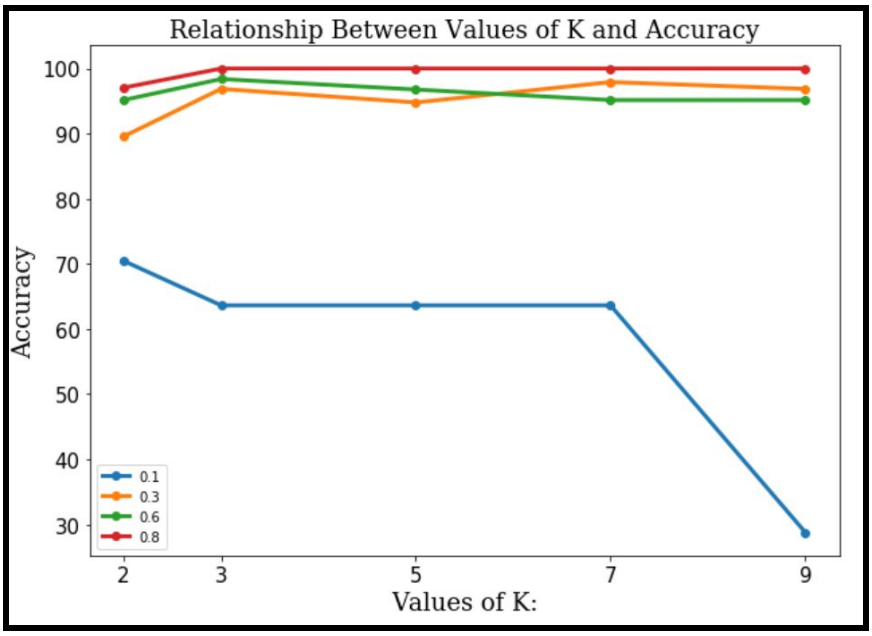
1. It runs through the whole dataset computing d between x and each training observation. We'll call the K points in the training data that are closest to x the set A
2. It then estimates the conditional probability for each class, that is, the fraction of points in A with that given class label. (Note I(x) is the indicator function which evaluates to 1 when the argument x is true and otherwise)
3. Finally, our input x gets assigned to the class with the largest probability.

To systematically vary the split, there are **4 split values → 0.1, 0.3, 0.6, 0.9**
For each split value, these are the 5 k values → **2, 3, 5, 7, 9**
Following are the number of elements in the train set and test set for each split respectively.

```
For split_value - 0.1
Train set: 17
Test set: 132
For split_value - 0.3
Train set: 53
Test set: 96
For split_value - 0.6
Train set: 87
Test set: 62
For split_value - 0.8
Train set: 115
Test set: 34
```

Following is the resultant dataframe and line plot obtained for the relationship between values of k and accuracy for each split values.

	0.1	0.3	0.6	0.9
2	94.029851	96.363636	94.827586	100.0
3	91.044776	95.454545	96.551724	100.0
5	88.805970	93.636364	96.551724	100.0
7	83.582090	93.636364	98.275862	100.0
9	64.179104	93.636364	96.551724	100.0



Here,

For blue split line - 10% data - training set || 90% data - test set

From the plot obtained above, it is evident that lesser the split percentage (which means less train data), less is the accuracy of model to classify data (as compared to other splits).

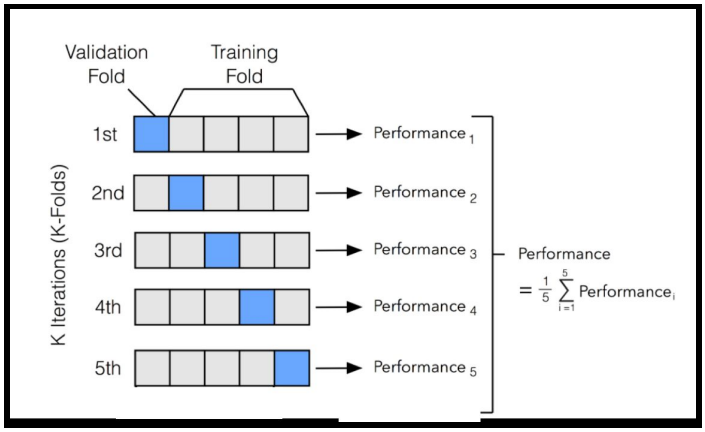
For red split line - 90% data - training set || 10% data - test set

Here, the model almost correctly classifies all the values in test data. (This can be because of overfitting as well)

From the above points, we can come to the following conclusions -

1. The accuracy increases when we provide more training data as compared to test data to the model.
2. As the 'k' in k-neighbors increases, the overall accuracy decreases (after a point). This happens because there are more neighbors that are tagged alongside each other to satisfy the 'k' value in k-nn. This in turn results in underfitting of data. Here for the blue split line, we can see when k = 9, the accuracy decreases as compared to accuracy when k = 7.

Cross-validation is a method for estimating the prediction accuracy of a model. It is a technique used to protect against overfitting in a predictive model, particularly in a case where the amount of data may be limited. In cross-validation, you make a fixed number of folds (or partitions) of the data, run the analysis on each fold, and then average the overall error estimate.The naive way to understand K-fold cross validation is with the following pictorial representation-

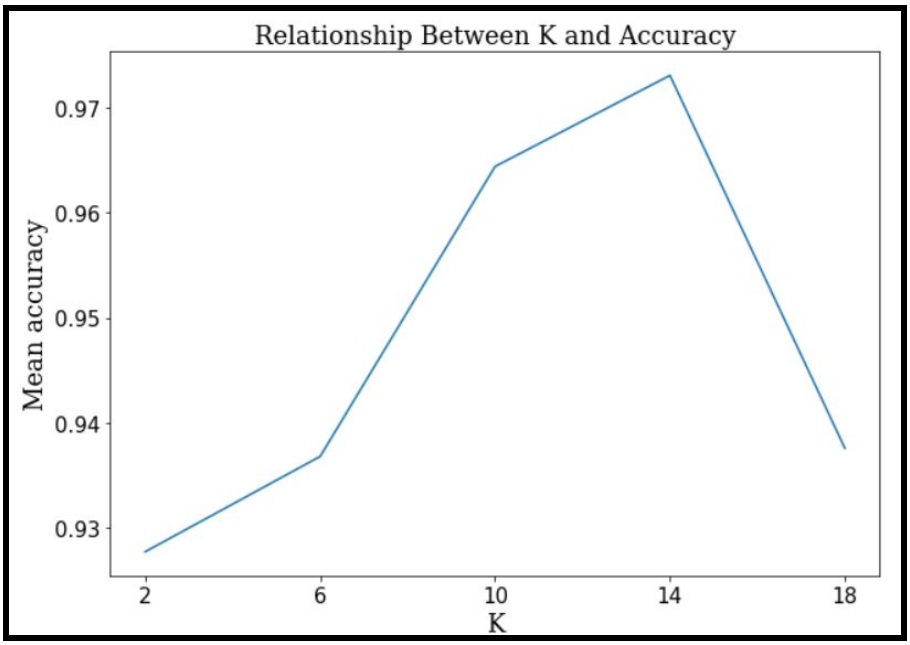


Overview of algorithm -

1. Divide dataset into k equal parts. → {k1, k2, k3, k4, … ,kn}
2. Isolate part k1 for test set → TestSet {k1} → y
3. Isolate remaining k-1 parts for train set → TrainSet {k2, k3, k4, k5} → X
4. Perform knn using cross-validation on the above configuration. →
   cross_val_score(KNN, X, y, no_of_folds)
5. Repeat the above process k times by isolating the train and test sets as mentioned above.
6. Remove the average summation of accuracy obtained while running the above steps k times.

Here, every point gets an equal opportunity to be represented in test set as compared to splitting the data in normal fashion where all the points are fixed to each of the train and test sets respectively. To implement k-fold cross validation, the KFold function from sklearn package is used. To calculate the accuracy, cross_eval_score() is used. The values of k in k-NN using cross validation is chosen as → [2, 6, 10, 14, 18]

Following is the result of running KNN using k-fold configuration with split/fold value as 5.



Here, x axis → values of k ; y axis → mean accuracy obtained for all the results for each k.
The accuracy increases when the k value in knn increases. As the data is split into multiple sets, there can be better results for the KNN accuracy in this configuration as compared to normal holdout method.

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.

Naive Bayes is a probabilistic machine learning model which is used as a classifier. The intuition behind this algorithm is Bayes theorem. Bayes theorem tells the probability of an event occurring given the probability of another event that has already occurred. Below is the mathematical equation of Bayes theorem.

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \rightarrow Posterior = \frac{likelihood \times prior}{evidence}$$

Bayes theorem states the relationship between joint distributions and conditional distributions.

$$p(x,y) = p(x).p(y|x) = p(y).p(x|y)$$

which means

$$p(x|y) = \frac{p(x).p(y|x)}{p(y)}$$

$$= \frac{p(x).p(y_1,\ldots,y_n|x_1,\ldots,x_m)}{p(y)}$$

$$= \frac{p(x)\prod_{i=1}^{n} p(y_i|y_1,\ldots,y_{i-1},x_1,\ldots,x_m)}{p(y)}$$

Naive Bayes is a simplification of Bayes theorem by assuming that ‘yi’ is conditionally independent of {y1,…,yi−1} given ‘x’.

$$p(x|y) = \frac{p(x)\prod_{i=1}^{n} p(y_i|x_1,\ldots,x_m)}{p(y)}$$

The name ‘naive’ is used because it assumes the features that go into the model is independent of each other. That is changing the value of one feature, does not directly influence or change the value of any of the other features used in the algorithm. For example, a fruit may be considered to be an apple if it is red, round, and about 4" in diameter. Even if these features depend on each other or upon the existence of the other features, a naive Bayes classifier considers all of these properties to independently contribute to the probability that this fruit is an apple. Basically, it's "naive" because it makes assumptions that may or may not turn out to be correct.

From the dataset provided, following are the new features extracted -

1. First two characters and last two characters - Extract first two and last two characters for each word.
2. From middle to end - Extract characters from middle to end for each word.

Following are the functions that define above features -

```
def get_from_mid(word):
    return {'from_mid_to_end' : word[len(word)//2 : len(word) - 1]}

def get_first2_last2(word):
    return {'first_word': word[0:1], 'last_word' : word[-2:]}
```

When the nltk naive bayes classifier is run with the above feature definitions, following are the observations -

get_first2_last2() →

```
------ First 2 last 2 ------
Cob is: male
John is: male
Barry is: male
Most Informative Features
               last_word = 'na'           female : male   =    100.1 : 1.0
               last_word = 'ia'           female : male   =     39.7 : 1.0
               last_word = 'ra'           female : male   =     37.1 : 1.0
               last_word = 'sa'           female : male   =     34.1 : 1.0
               last_word = 'ta'           female : male   =     33.8 : 1.0
0.82
```

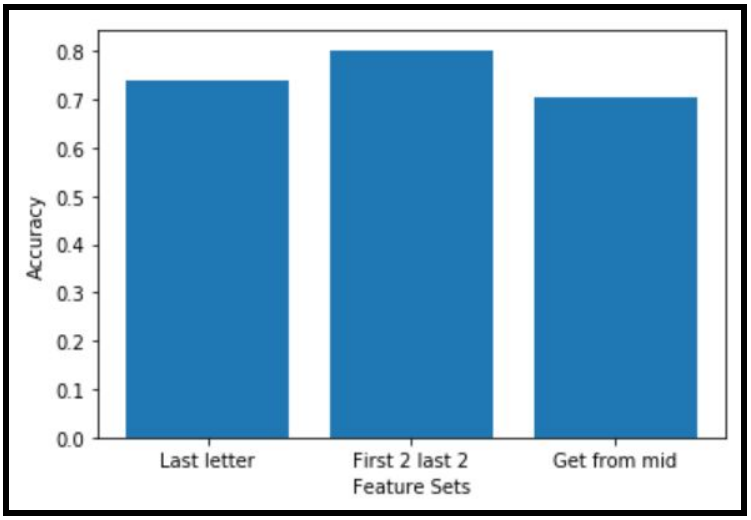
Here, we can see that the algorithm correctly classifies **'cob', 'john' and 'barry' as male**. Also, the algorithm produces **82% accuracy** with the above feature. The most information features also show that the last_word (last 2 characters) is more prominent than first_word. From the most informative feature, we can observe that for words having last_word as **'na'** there is a higher chance that it will be a female.

get_from_mid() →

```
------ Get from mid ------
Cob is: male
John is: male
Barry is: female
Most Informative Features
         from_mid_to_end = 'ett'          female : male   =     36.1 : 1.0
         from_mid_to_end = 'ss'           female : male   =     16.9 : 1.0
         from_mid_to_end = 'ann'          female : male   =     16.9 : 1.0
         from_mid_to_end = 'do'             male : female =     15.1 : 1.0
         from_mid_to_end = 'wi'             male : female =     14.0 : 1.0
0.742
```

Here, we can see that the algorithm incorrectly classifies 'barry' as female. The algorithm also predicts the lowest accuracy (74%). This in turn infers that the middle portion of each word doesn't act as a good feature in this classification. Here we can infer for words with middle portion, **'ett'** there is a higher chance that it will be a female. Similarly for words with middle portion, **'do'** there is a higher chance that it will be a male.

Following is the comparative plot of accuracy of existing feature vs newly selected features.

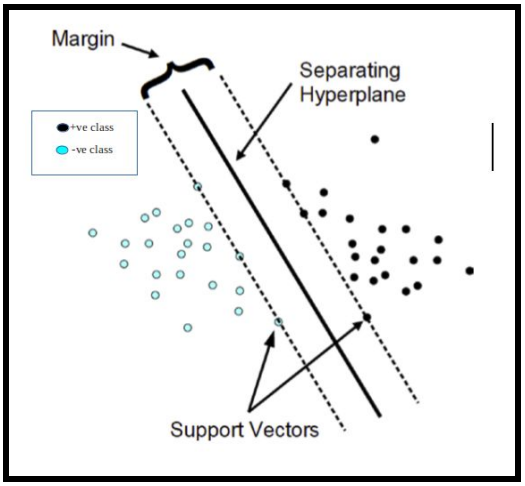


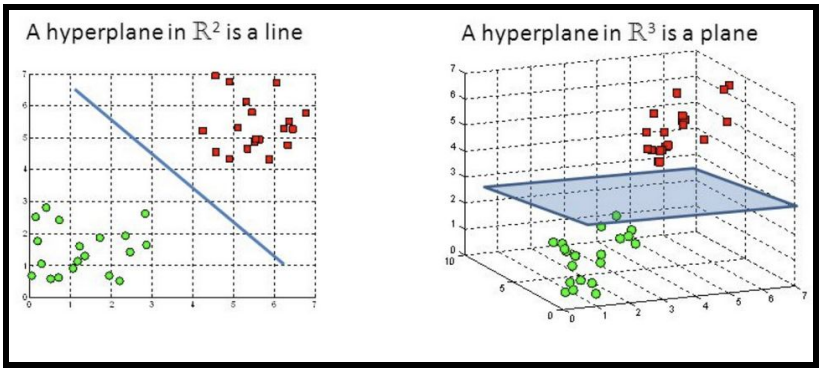
From the above results, we can infer that randomly adding features to classify data or increasing the length of featuresets doesn't improve the accuracy of Naive Bayes in all the scenarios. Adding meaningful features can result in better accuracy. (Here, the first2_last_2 feature worked better because it was more sensible to predict gender from first and last characters of each word than predicting gender from the mid).

Q3.) SVMs

So, this is just a quick use of an SVM; to show you how easy it is. It involves learning digit recognition. Now run three different training configurations and then test the outputs for 5-10 different digits; report results.

The SVM algorithm finds the points closest to the line from both the classes.These points are called the support vectors. The next step is to compute the distance between the line and the support vectors. This distance is called the margin. The primary goal is to maximize the margin. Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. The hyperplane for which the margin is maximum is the optimal hyperplane. Thus SVM tries to make a decision boundary in such a way that the separation between the two classes(that street) is as wide as possible.





Running SVM under different training configuration implies tuning the parameters of SVM to perform classification. In the given code, we have used the parameters gamma and C.

C →

The C parameter controls the tradeoff between classification of training points accurately and a smooth decision boundary. It suggests the model to choose data points as a support vector. If the value of C is

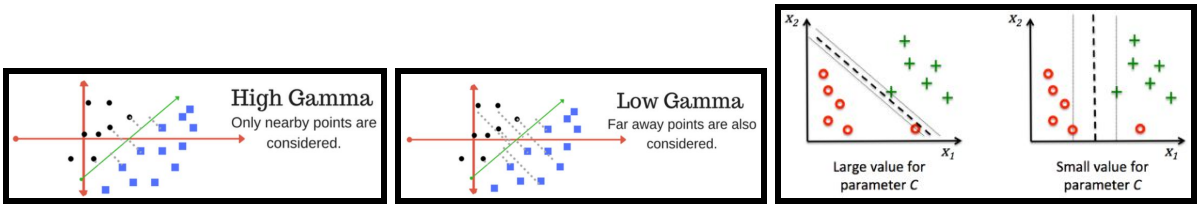
large then model choose more data points as a support vector and we get the higher variance and lower bias, which may lead to the problem of overfitting. The parameter C controls the trade off between smooth decision boundary and classifying training points correctly. A smaller C will allow more errors and margin errors and usually produce a larger margin. When C goes to Infinity, svm becomes a hard-margin.

Low C → allow more outliers || High C → allow fewer outliers

Gamma → It defines how far the influence of a single training example reaches.

When gamma has a very high value, then the decision boundary is just going to be dependent upon the points that are very close to the line which effectively results in ignoring some of the points that are very far from the decision boundary. This is because the closer points gain more weight and it results in a wiggly curve.On the other hand, if the gamma value is low even the far away points get considerable weight and we get a more linear curve.

Low gamma value → every point has far reach || High gamma value → every point has close reach.



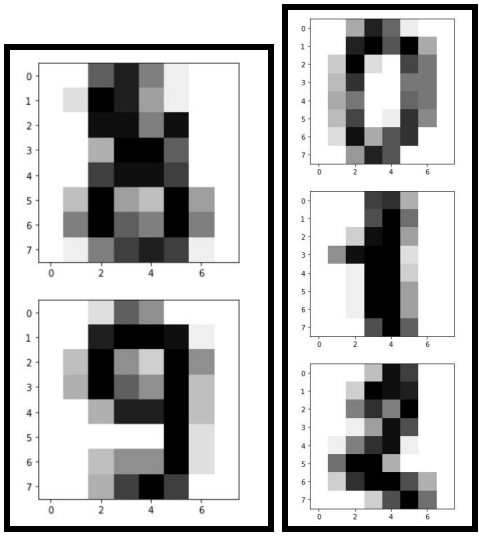
Following are the results when different training configurations are used.

values of gamma used → 0.0001, 1, 50000

values of C used → 0.01, 100, 10000000000000000

The above values are chosen with an intention to understand how the algorithm underfits the data for low gamma and low C. Similarly, how the algorithm overfits data for high gamma and high C.

Following are the actual images → These images resemble the digits -- 8, 9, 0, 1, 2



For each of the gamma and C configuration; we call predict() for each of the above images. Following is the result -

```
Gamma => 0.0001 || C => 0.01
Prediction: [3]
Prediction: [3]
Prediction: [3]
Prediction: [3]
Prediction: [3]
Gamma => 1 || C => 100
Prediction: [3]
Prediction: [9]
Prediction: [0]
Prediction: [1]
Prediction: [2]
Gamma => 50000 || C => 10000000000000000
Prediction: [3]
Prediction: [9]
Prediction: [0]
Prediction: [1]
Prediction: [2]
```

From the results following are the inferences -

1. When gamma is very low(0.0001), we can clearly see that the algorithm incorrectly predicts each of the digits as ‘3’. One of the probable reasons for this result is that algorithm has underfit the data.
2. However, for larger values of gamma (G => 1, G => 5000) and C(C => 100, C => 10000000000000000), the algorithm only incorrectly classifies digit ‘8’ as ‘3’.