

Text Analytics Tutorial 6

Question 1. K-NN

Plot Accuracy for Parameters

I chose values of 1, 5, 9, 13 and 17 for 'k' in this question and used ten evenly spaced training splits from 0-1 for 'Split'. To show these two dimensions alongside accuracy (total of three dimensions) I represented 'k' and 'Split' on the standard x y plane with accuracy represented by colour and opacity.



The results are somewhat surprising in that a 'k' value of 1 seems to get very high accuracy results. The worst results came from a split of 0.1 and 'k' values of 9, 13 and 17. This is not surprising considering that the training data is small, and this makes it hard to train accurately. Also with this dataset 'k' could be unreliable at such high values because there are many classes for the target feature and not a lot of data, it makes sense that 'k' would need to be kept small since the clusters of classes would also be relatively small.

Describe Algorithm for 5-Fold Cross Validation

I chose to implement a k-fold cross validation. The data is evenly split into 5 sections and in each case the majority of data is used for training and the last fifth for testing. To do this I modified the function *loadDataset* as seen below.

```
def loadDataset(filename, split):
    global trainingSet
    global testSet
    with open(filename, 'r') as csvfile:
        lines = csv.reader(csvfile)
        dataset = list(lines)
        for x in range(len(dataset) - 1):
            for y in range(4):
                dataset[x][y] = float(dataset[x][y])
    n = len(dataset)
    # Use value of i in main function to partition dataset each iteration
    if split == 1:
        testSet = dataset[:int(n / 5)]
        trainingSet = dataset[int(n / 5):]
    elif split == 2:
        testSet = dataset[int(n / 5):int((n / 5) * 2)]
        trainingSet = dataset[:int(n / 5)] + dataset[int((n / 5) * 2):]
    elif split == 3:
```

```

        testSet = dataset[int(n / 5) * 2:int(n / 5) * 3]
        trainingSet = dataset[:int(n / 5) * 2] + dataset[int(n / 5) * 3:]
    elif split == 4:
        testSet = dataset[int(n / 5) * 3:int(n / 5) * 4]
        trainingSet = dataset[:int(n / 5) * 3] + dataset[int(n / 5) * 4:]
    elif split == 5:
        testSet = dataset[-int(n / 5):]
        trainingSet = dataset[:-int(n / 5)]

for i in range(5):
    i+=1
    main(1, i)

```

I chose a value of k=1 because this yielded good results for a split of 0.2.

The results were as follows:

```

Accuracy: 100.0%
Accuracy: 100.0%
Accuracy: 86.66666666666667%
Accuracy: 93.33333333333333%
Accuracy: 83.33333333333334%

```

Overall the accuracy was 92.66%

Question 2. Bayes Classifiers

For my modification of the *gender_features* function I decided to do a similarly simple approach by just taking the last two letters of a person's name to try predict their gender, the code for which is shown below.

```

def gender_features(word):
    return {'last_pair_letters': word[-2] + word[-1]}

```

The accuracy of this classifier was better than just taking the last letter.

Function	Accuracy
<i>Last_letter</i>	76.8%
<i>Last_pair_letters</i>	77.4%

The most informative features from the different classifiers were as follows.

Most Informative Features for last letter classifier:

```

last_letter = 'a'      female : male = 38.5 : 1.0
last_letter = 'k'      male : female = 32.5 : 1.0
last_letter = 'f'      male : female = 17.2 : 1.0
last_letter = 'p'      male : female = 10.5 : 1.0
last_letter = 'v'      male : female = 10.5 : 1.0

```

Most informative features for last pair of letters classifier:

last_pair_letters = 'na' female : male = 98.5 : 1.0
 last_pair_letters = 'la' female : male = 77.0 : 1.0
 last_pair_letters = 'ia' female : male = 39.7 : 1.0
 last_pair_letters = 'us' male : female = 38.5 : 1.0
 last_pair_letters = 'rd' male : female = 31.3 : 1.0










It is not surprising that taking the last pair of letters would be slightly more informative for the classifier since male and female names tend to end in specific infixes which are captured better with more context (more than one letter).

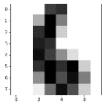
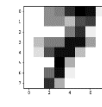
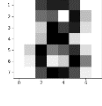
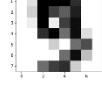
For example, looking at the names which end in 'na' it is 98.5 times for likely to be a female rather than a male which is an enormous difference. By comparison, the largest such example for single letters is those names which end in 'a', which are 38.5 times more likely to be female than male.

Question 3. SVMs

Here I decided to vary *gamma* and *c* whilst leaving the *kernel* parameter set to 'rbf'. The results of the iterations were as follows for 10 digits with 3 different configurations.

Gamma	C	Accuracy
0.001	100	90%
0.01	10	50%
0.001	10	90%

C	Gamma	Prediction	Value	Image
100	0.001	[0]	0	
10	0.01	[0]	0	
10	0.001	[0]	0	
100	0.001	[1]	1	
10	0.01	[1]	1	
10	0.001	[1]	1	
100	0.001	[2]	2	
10	0.01	[8]	2	
10	0.001	[2]	2	
100	0.001	[3]	3	
10	0.01	[3]	3	
10	0.001	[3]	3	
100	0.001	[4]	4	
10	0.01	[8]	4	
10	0.001	[4]	4	
100	0.001	[9]	5	
10	0.01	[9]	5	
10	0.001	[9]	5	

100	0.001	[6]	6	
10	0.01	[6]	6	
10	0.001	[6]	6	
100	0.001	[7]	7	
10	0.01	[8]	7	
10	0.001	[7]	7	
100	0.001	[8]	8	
10	0.01	[8]	8	
10	0.001	[8]	8	
100	0.001	[9]	9	
10	0.01	[8]	9	
10	0.001	[9]	9	

The code which I used for my iterations was the following:

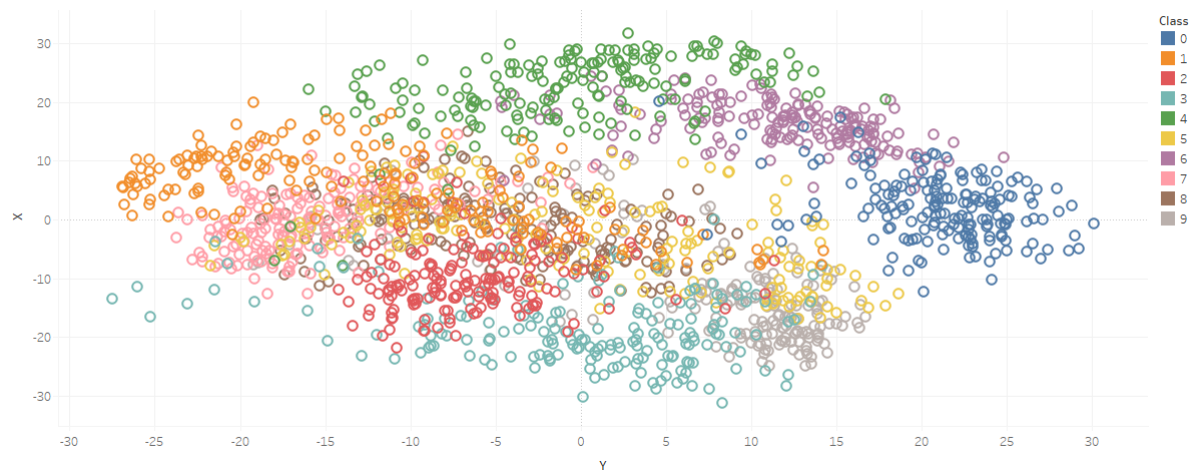
```
# Iterate configs
for i in range(3):
    # Config 1
    if i == 0:
        gamma = 0.001
        c = 100
    # Config 2
    elif i == 1:
        gamma = 0.01
        c = 10
    # Config 3
    elif i == 2:
        gamma = 0.001
        c = 10
    # Iterate digits
    for j in range(1, 11):
        digits = datasets.load_digits()
        classifier = svm.SVC(gamma=gamma, C=c, kernel='rbf')
        x, y = np.concatenate([digits.data[:j]] + [digits.data[j+1:]]),
np.concatenate([digits.target[:j]] + [digits.target[j+1:]])
        classifier.fit(x, y)
        # Make dataframe lists for CSV File
        c_list.append(c)
        gamma_list.append(gamma)
        actual_digit_list.append(digits.target[j])
        predicted_digit_list.append(classifier.predict(digits.data[j]))
```

This gave me several lists which I converted to a *Pandas DataFrame* object and used to plot the tables above.

The results suggest that the most important parameter here is *Gamma* rather than *C*. *Gamma* represents the reach of each data point regarding how it effects the SVM's decision boundary, *C* is a way of generalising the model. A higher value of *Gamma* seems to negatively impact the model, what this would seem to imply is that the data arrays which represent the image of each number are not very dissimilar, and that allowing a further reach for each number makes the results spread undesirably into each of the digit classes in their n-dimensional vector space.

The data for each part of each number array varies from 0-16, and each array is 64 numbers in length. This is not a particularly large variance and given the discrepancy between hand written digits it is not surprising that a lower *Gamma* parameter would give better results.

PCA Plot Of Digits

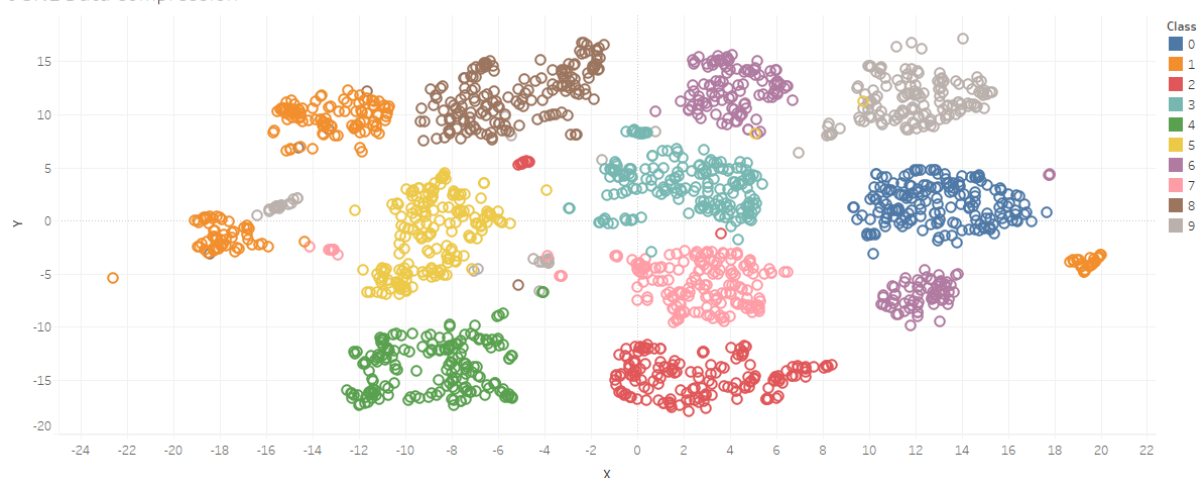


To finalise my analysis, I performed a principal component analysis of the dataset and visualised the results in 2 dimensions (plus colour dimension). As can be seen, the results were promising but (provided this visualisation is informative) it can clearly be seen how certain numbers (when drawn badly perhaps) become more ambiguous towards the centre of the plot. However, most of the data is (presumably) drawn more uniformly and is easily classified far from the other classes.

With this tight cluster of data in the middle of the plot, it also helps explain why a lower *Gamma* would produce better classification accuracy.

As an aside, I also utilised Google's *t-SNE* visualisation tool to compare its results to PCA. The results showed a much better clustering effect and it can easily be seen that there are distinct clusters in the data.

t-SNE Data Compression



In the future it will be necessary to utilise several data dimensionality reduction techniques to find the most appropriate one for the current task.