In [1]:
```
# Loading the required libraries:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cross_validation import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn import cross_validation
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model_selection module into w
hich all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.
20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:
```
from time import time

# Some helper functions:
def get_shape(seq):
    if type(seq) == type([]):
        print("The shape of data is:", len(seq),",",len(seq[0]))
    else:
        print("The shape of data is:", seq.shape)
    return

def time_taken(start):
    print("\nRuntime:", round(time()-start, 2), "seconds")
    return
```

4. Machine Learning Models

We will apply two ML Algorithms and generate 2 models:

1. Random Modelling Algorithm on a Sample of Quora Question Pairs Data.
2. K-Nearest Neighbour Algorithm on a Sample of Quora Question Pairs Data.

4.1 Loading Data

In [3]:
```
st = time()
# Load only a sample of the final features data:
quora_df = pd.read_csv('./final_features_100k.csv', nrows=25000)

# Data Info:
print(type(quora_df))
get_shape(quora_df)
time_taken(st)
quora_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
The shape of data is: (25000, 797)

Runtime: 7.92 seconds
```

Out[3]:

	Unnamed: 0	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	...	374_y	375_y	376_y	377_y	378_y	379_y	380_y	381_y	382_
0	0	0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	...	16.188503	33.233713	6.971700	-14.820828	15.534945	8.205955	-25.256606	1.552828	1.651827
1	1	1	0	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	...	-4.432317	-4.367793	41.101273	-0.930737	-15.686246	-7.275999	2.756560	-7.351970	3.10377
2	2	2	0	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	...	8.264448	-2.244750	11.084606	-16.741266	14.854023	15.726977	-1.298039	14.340431	11.66901
3	3	3	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	...	3.488654	3.906499	13.387563	-6.640244	6.378005	6.028185	2.511873	-3.830347	5.42107
4	4	4	0	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	...	-2.440844	11.887040	8.019029	-15.028031	8.280575	1.703147	-6.503707	11.263387	11.55681

5 rows × 797 columns

In [4]:
```
quora_df.tail()
```

Out[4]:

	Unnamed: 0	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	...	374_y	375_y	376_y	377_y	378_y	379_y	380_y	381_y	
24995	24995	24995	0	0.999950	0.666644	0.999950	0.499988	0.999975	0.571420	1.0	...	7.164913	23.039302	5.387981	0.075391	5.558455	-2.099565	-2.638070	-7.089084	-5.3
24996	24996	24996	0	0.666644	0.666644	0.799984	0.799984	0.749991	0.749991	0.0	...	2.033602	15.150122	6.668069	-2.651128	13.367594	5.552914	-11.574743	0.417787	7.59
24997	24997	24997	0	0.599988	0.428565	0.000000	0.000000	0.299997	0.299997	0.0	...	-2.951352	0.726570	5.036810	-6.168143	-8.204763	6.214043	-4.996871	5.989402	12.0
24998	24998	24998	1	0.499988	0.399992	0.499975	0.199996	0.499992	0.299997	0.0	...	9.845878	24.765205	5.713985	-8.277235	14.101339	3.312259	-7.348258	4.242257	22.6
24999	24999	24999	0	0.999967	0.749981	0.999983	0.857131	0.999989	0.818174	0.0	...	5.514413	2.062461	2.855732	-5.723221	-0.315934	8.459152	-7.826173	3.594841	1.02

5 rows × 797 columns

In [5]:
```
# We will drop some useless features:
y_class = quora_df['is_duplicate']
quora_df.drop(['id', 'is_duplicate', 'Unnamed: 0'], axis=1, inplace=True)
quora_df.head()
```

Out[5]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_diff	mean_len	...	374_y	375_y	376_y	377_y	378_y	379_y	380_y	381_
0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	0.0	13.0	...	16.188503	33.233713	6.971700	-14.820828	15.534945	8.205955	-25.256606	1.55282
1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	0.0	12.5	...	-4.432317	-4.367793	41.101273	-0.930737	-15.686246	-7.275999	2.756560	-7.35197
2	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0	0.0	12.0	...	8.264448	-2.244750	11.084606	-16.741266	14.854023	15.726977	-1.298039	14.3404
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	12.0	...	3.488654	3.906499	13.387563	-6.640244	6.378005	6.028185	2.511873	-3.83034
4	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	0.0	10.0	...	-2.440844	11.887040	8.019029	-15.028031	8.280575	1.703147	-6.503707	11.2633

5 rows × 794 columns

In [6]:
```
# We have our class variable as:
y_class.head()
```

Out[6]:
```
0    0
1    0
2    0
3    0
4    0
Name: is_duplicate, dtype: int64
```

In [8]:
```
type(y_class)
```

Out[8]:
```
pandas.core.series.Series
```

4.2 Converting strings to numerics

In [7]:
```python
st = time()

cols = list(quora_df.columns)
for i in cols:
    quora_df[i] = quora_df[i].apply(pd.to_numeric)
    print(i, end=", ")
```

```
cwc_min, cwc_max, csc_min, csc_max, ctc_min, ctc_max, last_word_eq, first_word_eq, abs_len_diff, mean_len, token_set_ratio, token_sort_ratio, fuzz_ratio, fuzz_parital_ratio, longest_s
ubstr_ratio, freq_qid1, freq_qid2, q1len, q2len, q1_n_words, q2_n_words, word_Common, word_Total, word_share, freq_q1+q2, freq_q1-q2, 0_x, 1_x, 2_x, 3_x, 4_x, 5_x, 6_x, 7_x, 8_x, 9_x,
10_x, 11_x, 12_x, 13_x, 14_x, 15_x, 16_x, 17_x, 18_x, 19_x, 20_x, 21_x, 22_x, 23_x, 24_x, 25_x, 26_x, 27_x, 28_x, 29_x, 30_x, 31_x, 32_x, 33_x, 34_x, 35_x, 36_x, 37_x, 38_x, 39_x, 40_
x, 41_x, 42_x, 43_x, 44_x, 45_x, 46_x, 47_x, 48_x, 49_x, 50_x, 51_x, 52_x, 53_x, 54_x, 55_x, 56_x, 57_x, 58_x, 59_x, 60_x, 61_x, 62_x, 63_x, 64_x, 65_x, 66_x, 67_x, 68_x, 69_x, 70_x,
71_x, 72_x, 73_x, 74_x, 75_x, 76_x, 77_x, 78_x, 79_x, 80_x, 81_x, 82_x, 83_x, 84_x, 85_x, 86_x, 87_x, 88_x, 89_x, 90_x, 91_x, 92_x, 93_x, 94_x, 95_x, 96_x, 97_x, 98_x, 99_x, 100_x, 10
1_x, 102_x, 103_x, 104_x, 105_x, 106_x, 107_x, 108_x, 109_x, 110_x, 111_x, 112_x, 113_x, 114_x, 115_x, 116_x, 117_x, 118_x, 119_x, 120_x, 121_x, 122_x, 123_x, 124_x, 125_x, 126_x, 127
_x, 128_x, 129_x, 130_x, 131_x, 132_x, 133_x, 134_x, 135_x, 136_x, 137_x, 138_x, 139_x, 140_x, 141_x, 142_x, 143_x, 144_x, 145_x, 146_x, 147_x, 148_x, 149_x, 150_x, 151_x, 152_x, 153_
x, 154_x, 155_x, 156_x, 157_x, 158_x, 159_x, 160_x, 161_x, 162_x, 163_x, 164_x, 165_x, 166_x, 167_x, 168_x, 169_x, 170_x, 171_x, 172_x, 173_x, 174_x, 175_x, 176_x, 177_x, 178_x, 179_
x, 180_x, 181_x, 182_x, 183_x, 184_x, 185_x, 186_x, 187_x, 188_x, 189_x, 190_x, 191_x, 192_x, 193_x, 194_x, 195_x, 196_x, 197_x, 198_x, 199_x, 200_x, 201_x, 202_x, 203_x, 204_x, 205_
x, 206_x, 207_x, 208_x, 209_x, 210_x, 211_x, 212_x, 213_x, 214_x, 215_x, 216_x, 217_x, 218_x, 219_x, 220_x, 221_x, 222_x, 223_x, 224_x, 225_x, 226_x, 227_x, 228_x, 229_x, 230_x, 231_
x, 232_x, 233_x, 234_x, 235_x, 236_x, 237_x, 238_x, 239_x, 240_x, 241_x, 242_x, 243_x, 244_x, 245_x, 246_x, 247_x, 248_x, 249_x, 250_x, 251_x, 252_x, 253_x, 254_x, 255_x, 256_x, 257_
x, 258_x, 259_x, 260_x, 261_x, 262_x, 263_x, 264_x, 265_x, 266_x, 267_x, 268_x, 269_x, 270_x, 271_x, 272_x, 273_x, 274_x, 275_x, 276_x, 277_x, 278_x, 279_x, 280_x, 281_x, 282_x, 283_
x, 284_x, 285_x, 286_x, 287_x, 288_x, 289_x, 290_x, 291_x, 292_x, 293_x, 294_x, 295_x, 296_x, 297_x, 298_x, 299_x, 300_x, 301_x, 302_x, 303_x, 304_x, 305_x, 306_x, 307_x, 308_x, 309_
x, 310_x, 311_x, 312_x, 313_x, 314_x, 315_x, 316_x, 317_x, 318_x, 319_x, 320_x, 321_x, 322_x, 323_x, 324_x, 325_x, 326_x, 327_x, 328_x, 329_x, 330_x, 331_x, 332_x, 333_x, 334_x, 335_
x, 336_x, 337_x, 338_x, 339_x, 340_x, 341_x, 342_x, 343_x, 344_x, 345_x, 346_x, 347_x, 348_x, 349_x, 350_x, 351_x, 352_x, 353_x, 354_x, 355_x, 356_x, 357_x, 358_x, 359_x, 360_x, 361_
x, 362_x, 363_x, 364_x, 365_x, 366_x, 367_x, 368_x, 369_x, 370_x, 371_x, 372_x, 373_x, 374_x, 375_x, 376_x, 377_x, 378_x, 379_x, 380_x, 381_x, 382_x, 383_x, 0_y, 1_y, 2_y, 3_y, 4_y, 5
_y, 6_y, 7_y, 8_y, 9_y, 10_y, 11_y, 12_y, 13_y, 14_y, 15_y, 16_y, 17_y, 18_y, 19_y, 20_y, 21_y, 22_y, 23_y, 24_y, 25_y, 26_y, 27_y, 28_y, 29_y, 30_y, 31_y, 32_y, 33_y, 34_y, 35_y, 36_
y, 37_y, 38_y, 39_y, 40_y, 41_y, 42_y, 43_y, 44_y, 45_y, 46_y, 47_y, 48_y, 49_y, 50_y, 51_y, 52_y, 53_y, 54_y, 55_y, 56_y, 57_y, 58_y, 59_y, 60_y, 61_y, 62_y, 63_y, 64_y, 65_y, 66_y,
67_y, 68_y, 69_y, 70_y, 71_y, 72_y, 73_y, 74_y, 75_y, 76_y, 77_y, 78_y, 79_y, 80_y, 81_y, 82_y, 83_y, 84_y, 85_y, 86_y, 87_y, 88_y, 89_y, 90_y, 91_y, 92_y, 93_y, 94_y, 95_y, 96_y, 97_
y, 98_y, 99_y, 100_y, 101_y, 102_y, 103_y, 104_y, 105_y, 106_y, 107_y, 108_y, 109_y, 110_y, 111_y, 112_y, 113_y, 114_y, 115_y, 116_y, 117_y, 118_y, 119_y, 120_y, 121_y, 122_y, 123_y,
124_y, 125_y, 126_y, 127_y, 128_y, 129_y, 130_y, 131_y, 132_y, 133_y, 134_y, 135_y, 136_y, 137_y, 138_y, 139_y, 140_y, 141_y, 142_y, 143_y, 144_y, 145_y, 146_y, 147_y, 148_y, 149_y, 1
50_y, 151_y, 152_y, 153_y, 154_y, 155_y, 156_y, 157_y, 158_y, 159_y, 160_y, 161_y, 162_y, 163_y, 164_y, 165_y, 166_y, 167_y, 168_y, 169_y, 170_y, 171_y, 172_y, 173_y, 174_y, 175_y, 17
6_y, 177_y, 178_y, 179_y, 180_y, 181_y, 182_y, 183_y, 184_y, 185_y, 186_y, 187_y, 188_y, 189_y, 190_y, 191_y, 192_y, 193_y, 194_y, 195_y, 196_y, 197_y, 198_y, 199_y, 200_y, 201_y, 202
_y, 203_y, 204_y, 205_y, 206_y, 207_y, 208_y, 209_y, 210_y, 211_y, 212_y, 213_y, 214_y, 215_y, 216_y, 217_y, 218_y, 219_y, 220_y, 221_y, 222_y, 223_y, 224_y, 225_y, 226_y, 227_y, 228_
y, 229_y, 230_y, 231_y, 232_y, 233_y, 234_y, 235_y, 236_y, 237_y, 238_y, 239_y, 240_y, 241_y, 242_y, 243_y, 244_y, 245_y, 246_y, 247_y, 248_y, 249_y, 250_y, 251_y, 252_y, 253_y, 254_
y, 255_y, 256_y, 257_y, 258_y, 259_y, 260_y, 261_y, 262_y, 263_y, 264_y, 265_y, 266_y, 267_y, 268_y, 269_y, 270_y, 271_y, 272_y, 273_y, 274_y, 275_y, 276_y, 277_y, 278_y, 279_y, 280_
y, 281_y, 282_y, 283_y, 284_y, 285_y, 286_y, 287_y, 288_y, 289_y, 290_y, 291_y, 292_y, 293_y, 294_y, 295_y, 296_y, 297_y, 298_y, 299_y, 300_y, 301_y, 302_y, 303_y, 304_y, 305_y, 306_
y, 307_y, 308_y, 309_y, 310_y, 311_y, 312_y, 313_y, 314_y, 315_y, 316_y, 317_y, 318_y, 319_y, 320_y, 321_y, 322_y, 323_y, 324_y, 325_y, 326_y, 327_y, 328_y, 329_y, 330_y, 331_y, 332_
y, 333_y, 334_y, 335_y, 336_y, 337_y, 338_y, 339_y, 340_y, 341_y, 342_y, 343_y, 344_y, 345_y, 346_y, 347_y, 348_y, 349_y, 350_y, 351_y, 352_y, 353_y, 354_y, 355_y, 356_y, 357_y, 358_
y, 359_y, 360_y, 361_y, 362_y, 363_y, 364_y, 365_y, 366_y, 367_y, 368_y, 369_y, 370_y, 371_y, 372_y, 373_y, 374_y, 375_y, 376_y, 377_y, 378_y, 379_y, 380_y, 381_y, 382_y, 383_y,
```

In [8]:
```python
time_taken(st)
```

```
Runtime: 87.6 seconds
```

4.3 Random Train-Test Split (70:30)

In [9]:

```
# Our class variable is y_class:
X_train, X_test, y_train, y_test = train_test_split(
    quora_df, y_class, stratify=y_class, test_size=0.3
)

print('Number of data points in train data:', X_train.shape)
print('Number of data points in test data :', X_test.shape)
```

```
Number of data points in train data: (17500, 794)
Number of data points in test data : (7500, 794)
```

In [10]:

```
print(type(X_train))
X_train.tail()
```

```
<class 'pandas.core.frame.DataFrame'>
```

Out[10]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_diff	mean_len	...	374_y	375_y	376_y	377_y	378_y	379_y	380_y	38
14738	0.749981	0.599988	0.666644	0.666644	0.714276	0.624992	0.0	1.0	0.0	7.5	...	6.902403	11.493473	5.613606	-4.877812	13.043599	-1.132098	-4.538146	-3.3018
22866	0.749981	0.749981	0.749981	0.749981	0.749991	0.749991	1.0	1.0	0.0	8.0	...	3.047488	2.449508	-5.921124	3.526991	10.510124	2.405635	-7.946949	2.6444
24591	0.499975	0.499975	0.999900	0.999900	0.666644	0.666644	1.0	1.0	0.0	3.0	...	1.473008	-2.153633	0.372134	3.989067	-2.932041	-1.881972	-0.737835	15.230
20393	0.499988	0.399992	0.599988	0.333330	0.555549	0.333331	0.0	0.0	0.0	12.0	...	-1.791243	13.849159	1.789727	-7.466151	11.052734	5.252021	-16.688383	4.6421
21438	0.749981	0.749981	0.749981	0.599988	0.749991	0.666659	0.0	1.0	0.0	8.5	...	4.836686	9.265958	4.292188	-4.779898	13.177334	9.700841	-5.874484	5.4856

5 rows × 794 columns

In [11]:

```
X_test.tail()
```

Out[11]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_diff	mean_len	...	374_y	375_y	376_y	377_y	378_y	379_y	380_y	38
18463	0.333322	0.333322	0.999975	0.999975	0.714276	0.714276	0.0	1.0	0.0	7.0	...	1.600818	8.941236	4.398032	-3.078369	13.817652	7.158675	-11.921655	7.7252
4039	0.999980	0.999980	0.999950	0.666644	0.999986	0.874989	1.0	1.0	0.0	7.5	...	0.609111	8.085004	7.153885	1.884657	-11.188039	1.521977	0.628922	2.9140
9761	0.666644	0.399992	0.749981	0.374995	0.714276	0.333331	0.0	0.0	0.0	11.0	...	6.810799	1.547490	3.335044	-3.722337	14.990891	8.770758	-7.148856	5.8169
12817	0.999967	0.499992	0.999950	0.399992	0.999980	0.454541	0.0	1.0	0.0	8.0	...	-5.335906	-3.942303	-5.758041	-1.118189	11.369048	-5.043033	-10.508012	-4.148
16519	0.142855	0.076922	0.499988	0.285710	0.272725	0.149999	0.0	0.0	0.0	15.5	...	-6.623911	-8.571070	3.896555	-2.758711	-9.771975	-3.437023	-14.918431	16.211

5 rows × 794 columns

In [12]:
```
get_shape(y_train)
y_train.tail()
```

```
The shape of data is: (17500,)
```

Out[12]:
```
14738    1
22866    0
24591    1
20393    1
21438    0
Name: is_duplicate, dtype: int64
```

In [13]:
```
get_shape(y_test)
y_test.tail()
```

```
The shape of data is: (7500,)
```

Out[13]:
```
18463    0
4039     1
9761     0
12817    1
16519    0
Name: is_duplicate, dtype: int64
```

In [14]:
```
# Now we will see the distribution of points classwise:
print("-"*10, "Distribution of O/P Variable in train data", "-"*10)
tr_disb = Counter(y_train)
print("Number of data points that correspond to 'is_duplicate = 0' are:", tr_disb[0])
print("Number of data points that correspond to 'is_duplicate = 1' are:", tr_disb[1])
tr_len = len(y_train)
print("Total Number of points in train:", tr_len, "\n")
print("O/P (or) class-label: 'is_duplicate'")
print("is_duplicate = 0:", float(tr_disb[0]/tr_len),
     "\nis_duplicate = 1:", float(tr_disb[1]/tr_len))
```

```
---------- Distribution of O/P Variable in train data ----------
Number of data points that correspond to 'is_duplicate = 0' are: 10986
Number of data points that correspond to 'is_duplicate = 1' are: 6514
Total Number of points in train: 17500

O/P (or) class-label: 'is_duplicate'
is_duplicate = 0: 0.6277714285714285
is_duplicate = 1: 0.3722285714285714
```

4.3 Building a Random Model

We will find the worst case accuracy score using a random model.

In [15]:
```
# We will create a list which has exactly same size as our test data
# and generate a non-uniform random model:
random_y = np.random.choice(2, size=len(y_test),
                            p = [0.1, 0.9]
                           )
random_y
```

Out[15]:
```
array([1, 1, 1, ..., 1, 0, 1], dtype=int64)
```

In [16]:
```
# Now we check the accuracy score:
rand_acc = accuracy_score(random_y, y_test, normalize=True) * float(100)
print("Accuracy Score for Random Model is {}%".format(rand_acc))
```

```
Accuracy Score for Random Model is 39.800000000000004%
```

With a Random Model, we are getting ~40% Accuracy, i.e., Our Random Model is able to predict whether 2 questions are similar or not, correctly, only 50% of the time. Therefore, this is the worst case Accuracy Score.

We want our k-NN to get an Accuracy Score > 40%.

4.4 Building k-Nearest Neighbours Model using Simple Cross Validation

In [17]:
```
# Split the train data into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = train_test_split(
    X_train, y_train, stratify=y_train, test_size=0.3
)

# train and cv data info:
print('Number of data points in train data:', X_tr.shape)
print('Number of data points in cross validation data :', X_cv.shape)
```

```
Number of data points in train data: (12250, 794)
Number of data points in cross validation data : (5250, 794)
```

```
In [18]: # Now we will see the distribution of points classwise:
print("-"*15, "Distribution of O/P Variable in train data", "-"*15)
train_tr_disb = Counter(y_tr)
print("Number of data points that correspond to 'is_duplicate = 0' are:",
      train_tr_disb[0])
print("Number of data points that correspond to 'is_duplicate = 1' are:",
      train_tr_disb[1])
train_tr_len = len(y_tr)
print("Total Number of points in train:", train_tr_len, "\n")
print("O/P (or) class-label: 'is_duplicate'")
print("is_duplicate = 0:", float(train_tr_disb[0]/train_tr_len),
     "\nis_duplicate = 1:", float(train_tr_disb[1]/train_tr_len))
```

```
--------------- Distribution of O/P Variable in train data ---------------
Number of data points that correspond to 'is_duplicate = 0' are: 7690
Number of data points that correspond to 'is_duplicate = 1' are: 4560
Total Number of points in train: 12250

O/P (or) class-label: 'is_duplicate'
is_duplicate = 0: 0.6277551020408163
is_duplicate = 1: 0.3722448979591837
```

Hyper Parameter Selection (or) Selection of Optimal K

```
In [20]: # Finding the right k and applying k-NN using simple cross-validation:
# Hyper parameter selection:

# Creating odd list of K for K-NN:
my_list = list(range(0,100))
neighbours = list(filter(lambda x: x%2 != 0, my_list))
print("We will test K-NN Algorithm for these values of K:\n")
for i in neighbours:
    print(i, end=' ')
```

```
We will test K-NN Algorithm for these values of K:

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

In [21]:

```
# Now we have all the odd numbers, we can now apply the sklearn
# implementation of KNN to know the similarity/polarity between two questions:

st = time()

# Code for hyper parameter selection:
for k in neighbours:

    # Configured parameters are:-
    #
    # 1. algorithm = 'auto':
    #    automatically choose the algorithm (KDTree, BallTree or Brute Force)
    #
    # 2. metric = 'minkowski', p = 2:
    #    Use L2 Minkowski Distance which is nothing but Euclidean Distance.
    #
    # 3. n_jobs = -1:
    #    Use all the CPU cores to apply KNN Classfication.

    # Instantiate the learning model:
    knn = KNeighborsClassifier(
        n_neighbors = k,
        algorithm = 'auto',
        metric = 'minkowski',
        p = 2,
        n_jobs = -1
    )

    # Fitting the model on train:
    knn.fit(X_tr, y_tr)

    # Predict the response on cross validation:
    predict_y_cv = knn.predict(X_cv)

    # Evaluate the cross validation accuracy:
    acc = accuracy_score(predict_y_cv, y_cv, normalize=True) * float(100)
    print('\nCross Validation Accuracy for k={} is {}%'
         .format(k, acc))

time_taken(st)
```

```
Cross Validation Accuracy for k=1 is 61.96190476190476%

Cross Validation Accuracy for k=3 is 63.29523809523809%

Cross Validation Accuracy for k=5 is 63.21904761904762%

Cross Validation Accuracy for k=7 is 63.847619047619055%

Cross Validation Accuracy for k=9 is 64.07619047619048%

Cross Validation Accuracy for k=11 is 64.11428571428571%

Cross Validation Accuracy for k=13 is 64.32380952380953%

Cross Validation Accuracy for k=15 is 64.36190476190477%

Cross Validation Accuracy for k=17 is 64.4%

Cross Validation Accuracy for k=19 is 64.64761904761905%

Cross Validation Accuracy for k=21 is 64.72380952380952%

Cross Validation Accuracy for k=23 is 64.47619047619048%

Cross Validation Accuracy for k=25 is 64.64761904761905%

Cross Validation Accuracy for k=27 is 65.16190476190476%

Cross Validation Accuracy for k=29 is 65.18095238095239%

Cross Validation Accuracy for k=31 is 64.91428571428571%

Cross Validation Accuracy for k=33 is 65.04761904761904%

Cross Validation Accuracy for k=35 is 64.62857142857142%

Cross Validation Accuracy for k=37 is 64.32380952380953%

Cross Validation Accuracy for k=39 is 64.13333333333333%

Cross Validation Accuracy for k=41 is 64.22857142857143%

Cross Validation Accuracy for k=43 is 64.03809523809524%

Cross Validation Accuracy for k=45 is 64.4%

Cross Validation Accuracy for k=47 is 64.53333333333333%

Cross Validation Accuracy for k=49 is 64.15238095238095%

Cross Validation Accuracy for k=51 is 64.51428571428572%

Cross Validation Accuracy for k=53 is 64.59047619047618%

Cross Validation Accuracy for k=55 is 64.4%

Cross Validation Accuracy for k=57 is 64.28571428571429%
```

```
Cross Validation Accuracy for k=59 is 64.24761904761904%

Cross Validation Accuracy for k=61 is 64.28571428571429%

Cross Validation Accuracy for k=63 is 64.41904761904762%

Cross Validation Accuracy for k=65 is 64.24761904761904%

Cross Validation Accuracy for k=67 is 64.03809523809524%

Cross Validation Accuracy for k=69 is 64.15238095238095%

Cross Validation Accuracy for k=71 is 63.98095238095238%

Cross Validation Accuracy for k=73 is 63.866666666666674%

Cross Validation Accuracy for k=75 is 63.82857142857142%

Cross Validation Accuracy for k=77 is 63.94285714285714%

Cross Validation Accuracy for k=79 is 64.05714285714285%

Cross Validation Accuracy for k=81 is 64.15238095238095%

Cross Validation Accuracy for k=83 is 64.11428571428571%

Cross Validation Accuracy for k=85 is 63.714285714285715%

Cross Validation Accuracy for k=87 is 63.904761904761905%

Cross Validation Accuracy for k=89 is 63.82857142857142%

Cross Validation Accuracy for k=91 is 63.67619047619048%

Cross Validation Accuracy for k=93 is 63.542857142857144%

Cross Validation Accuracy for k=95 is 63.714285714285715%

Cross Validation Accuracy for k=97 is 63.61904761904762%

Cross Validation Accuracy for k=99 is 63.88571428571429%

Runtime: 2152.39 seconds
```

Cross Validation Accuracy for k=29 is 65.18095238095239%. This is highest accuracy score out of all the accuracy scores.

Therefore, we got our k=29, i.e., we will consider the majority vote of the classes of 29 nearest neighbours in the vicinity of a query point -> xq.

Applying the K Value from Simple Cross Validation on Test Data

In [23]:
```
# Configured parameters are:-
#
# 1. algorithm = 'auto':
#    automatically choose the algorithm (KDTree, BallTree or Brute Force)
#
# 2. metric = 'minkowski', p = 2:
#    Use L2 Minkowski Distance which is nothing but Euclidean Distance.
#
# 3. n_jobs = -1:
#    Use all the CPU cores to apply KNN Classfication.

# Instantiate the learning model with k=29:
k_simple = 29
knn_simple_cv = KNeighborsClassifier(
    n_neighbors = k_simple,
    algorithm = 'auto',
    metric = 'minkowski',
    p = 2,
    n_jobs = -1
)

# Fitting the model on train data:
knn_simple_cv.fit(X_tr, y_tr)

# Predict the response on test data:
predict_y_test_simple_cv = knn_simple_cv.predict(X_test)

# Evaluate the test accuracy:
acc_test_simple = accuracy_score(predict_y_test_simple_cv, y_test, normalize=True) * float(100)
print('\n****** Test Accuracy for k={} is {}% *******'
     .format(k_simple, acc_test_simple))
```

```
****** Test Accuracy for k=29 is 64.62666666666667% *******
```

We will now apply K-NN using K-fold Cross Validation to get the best K, so that we can classify whether question1 is similar to question2 or not.

4.5 Building k-Nearest Neighbours Model using K-fold Cross Validation

Here, the k used for k-NN is a Hyper Parameter which tells us the number of neighbours that the algorithm is considering before making a decision about the class of a query point.

But, K used in K-fold Cross Validation is the number of folds/divisions we are making in our data, to consider the data as train and cross validation data with different division each time. After we get scores for each division, we take the mean of all of the scores, and that's our accuracy score of the k-fold cross validation.

K = 10: 10 fold Cross Validation

In [24]:
```python
# Empty list to store the cross validation scores:
cv_scores = []

st = time()
# Perform 10-fold cross validation:
for k in neighbours:
        # Configured parameters are:-
    #
    # 1. algorithm = 'auto':
    #    automatically choose the algorithm (KDTree, BallTree or Brute Force)
    #
    # 2. metric = 'minkowski', p = 2:
    #    Use L2 Minkowski Distance which is nothing but Euclidean Distance.
    #
    # 3. n_jobs = -1:
    #    Use all the CPU cores to apply KNN Classfication.

    # Instantiate the learning model:
    knn = KNeighborsClassifier(
        n_neighbors = k,
        algorithm = 'auto',
        metric = 'minkowski',
        p = 2,
        n_jobs = 3
    )

    # cv = 10: meaning 10 folds in the given data to get combinations
    # of train and cross validation data
    scores = cross_val_score(
        knn, X_train, y_train, cv=10, scoring='accuracy'
    )

    # record all the scores until now:
    cv_scores.append(scores.mean())
```

In [25]:
```python
time_taken(st)
```

```
Runtime: 13361.46 seconds
```

In [32]:
```python
for i in cv_scores:
    print(i, end=', ')
```

```
0.629255119617, 0.639312820543, 0.641598535053, 0.640970257035, 0.645256919435, 0.644399122845, 0.642683596016, 0.646397687043, 0.645140511216, 0.646170650141, 0.647427564818, 0.64697
0356804, 0.647999679272, 0.648456234785, 0.647771793651, 0.647599124002, 0.648113213611, 0.646570160102, 0.646456690733, 0.647198470493, 0.647997752536, 0.646625964463, 0.6462843481,
0.643712363698, 0.644169277965, 0.644398633366, 0.644455711029, 0.643769800489, 0.642684477891, 0.643713082066, 0.642055710246, 0.641712526273, 0.64256826544, 0.642283334754, 0.643255
318745, 0.642284118502, 0.644111580174, 0.642111840951, 0.641254110114, 0.640511709816, 0.642911742898, 0.643882681916, 0.642682943103, 0.642968265664, 0.643197196333, 0.643711873884,
0.641768689762, 0.641368493545, 0.641768950726, 0.640398142208,
```

In [33]:
```
# Changing to Misclassification error:
MSE = [1-x for x in cv_scores]

for i in MSE:
    print(i, end=', ')
```

```
0.370744880383, 0.360687179457, 0.358401464947, 0.359029742965, 0.354743080565, 0.355600877155, 0.357316403984, 0.353602312957, 0.354859488784, 0.353829349859, 0.352572435182, 0.35302
9643196, 0.352000320728, 0.351543765215, 0.352228206349, 0.352400875998, 0.351886786389, 0.353429839898, 0.353543309267, 0.352801529507, 0.352002247464, 0.353374035537, 0.3537156519,
0.356287636302, 0.355830722035, 0.355601366634, 0.355544288971, 0.356230199511, 0.357315522109, 0.356286917934, 0.357944289754, 0.358287473727, 0.35743173456, 0.357716665246, 0.356744
681255, 0.357715881498, 0.355888419826, 0.357888159049, 0.358745889886, 0.359488290184, 0.357088257102, 0.356117318084, 0.357317056897, 0.357031734336, 0.356802803667, 0.356288126116,
0.358231310238, 0.358631506455, 0.358231049274, 0.359601857792,
```

In [34]:
```
# Now, we will determine the best k:
optimal_k = neighbours[MSE.index(min(MSE))]
print("The optimal number of neighbours is:", optimal_k)
```

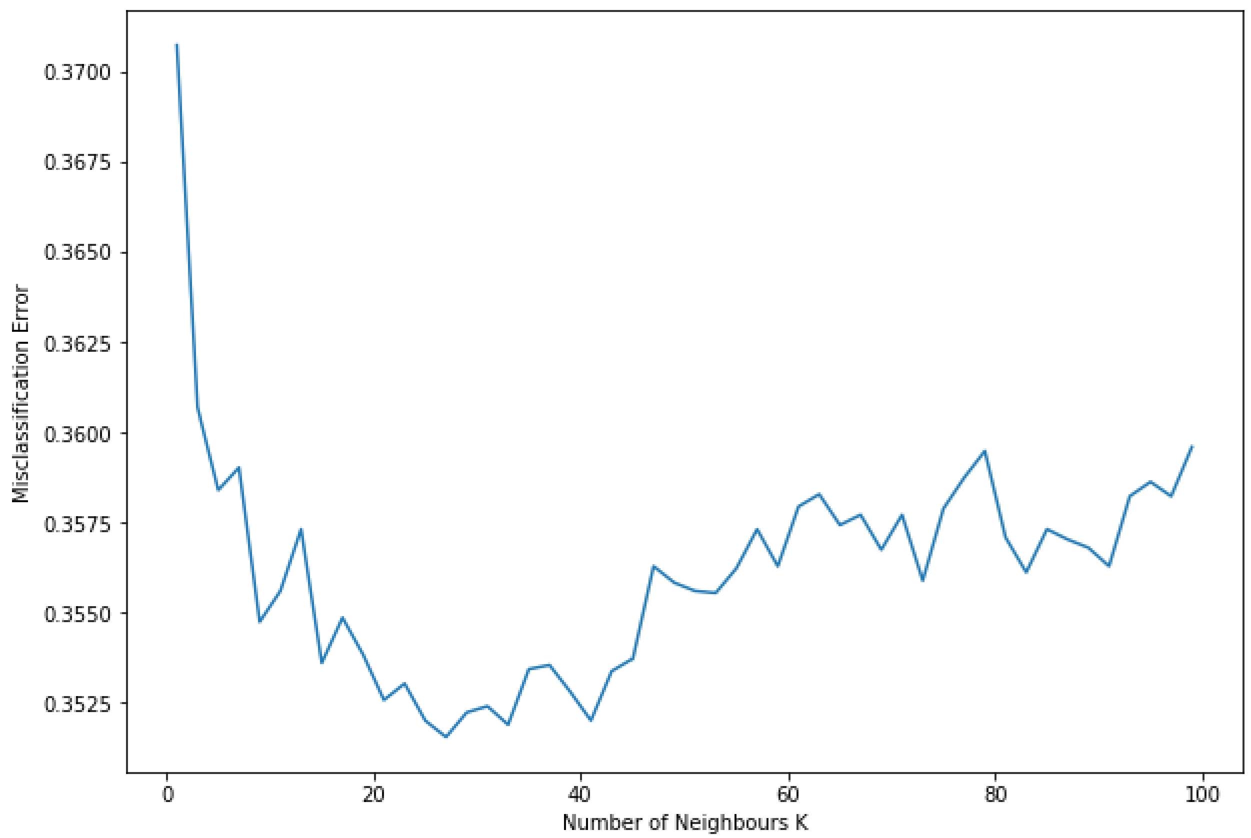
```
The optimal number of neighbours is: 27
```

In [37]:
```
# Plot the Misclassification Error v/s k:
plt.figure(figsize=(10,7))
plt.plot(neighbours, MSE)

#for xy in zip(neighbours, np.round(MSE, 2)):
    #plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

plt.xlabel('Number of Neighbours K')
plt.ylabel('Misclassification Error')
plt.show()

print("The Miscalssification Error for each k value is:\n", np.round(MSE, 3))
```




```
The Miscalssification Error for each k value is:
 [ 0.371  0.361  0.358  0.359  0.355  0.356  0.357  0.354  0.355  0.354
  0.353  0.353  0.352  0.352  0.352  0.352  0.352  0.353  0.354  0.353
  0.352  0.353  0.354  0.356  0.356  0.356  0.356  0.356  0.357  0.356
  0.358  0.358  0.357  0.358  0.357  0.358  0.356  0.358  0.359  0.359
  0.357  0.356  0.357  0.357  0.357  0.356  0.358  0.359  0.358  0.36 ]
```

From the plot above, we can see that the lowest value of Misclassification error is generated in between k=[20, 21, ..., 40]. That's the reason, we got our optimal_k to be 27.

Let us see the accuracy score after querying the k-NN model with the test data.

In [40]:
```
# KNN with k = optimal_k
st = time()
# Configured parameters are:-
#
# 1. algorithm = 'auto':
#    automatically choose the algorithm (KDTree, BallTree or Brute Force)
#
# 2. metric = 'minkowski', p = 2:
#    Use L2 Minkowski Distance which is nothing but Euclidean Distance.
#
# 3. n_jobs = -1:
#    Use all the CPU cores to apply KNN Classfication.

# Instantiate the learning model:
knn_optimal = KNeighborsClassifier(
    n_neighbors = optimal_k,
    algorithm = 'auto',
    metric = 'minkowski',
    p = 2,
    n_jobs = 3
)

# Fitting the model on train:
knn_optimal.fit(X_train, y_train)

# Predict the response on test:
predict_y_test = knn_optimal.predict(X_test)

# Evaluate the test accuracy:
acc_test = accuracy_score(predict_y_test, y_test, normalize=True) * float(100)
print('''\nThe Accuracy of k-NN classifier on Quora Question Pairs Dataset
for predicting whether two given questions have the same intent or not with
k={} is {}%'''.format(optimal_k, acc_test))

time_taken(st)
```

```
The Accuracy of k-NN classifier on Quora Question Pairs Dataset
for predicting whether two given questions have the same intent or not with
k=27 is 65.70666666666666%

Runtime: 111.65 seconds
```