In [1]:

```
import sys
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.datasets import load_svmlight_file
from sklearn import preprocessing
import pylab as pl
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:

```
def main(data, K_value, distance, vet_acur, vet_confusion):

        # loads data
        #print ("Loading data...")
        X_data, y_data = load_svmlight_file(data)
        # splits data
        #print ("Spliting data...")
        X_train, X_test, y_train, y_test =  train_test_split(X_data, y_data, test_size=0.5,
        # x vetor de características e Y labels
        X_train = X_train.toarray()
        X_test = X_test.toarray()

        # fazer a normalizacao dos dados #######
        #scaler = preprocessing.MinMaxScaler()
        #X_train = scaler.fit_transform(X_train_dense)
        #X_test = scaler.fit_transform(X_test_dense)

        # cria um kNN
        neigh = KNeighborsClassifier(n_neighbors=K_value, metric=distance)
        #treinamento
        #print ('Fitting knn')
        neigh.fit(X_train, y_train)

        # predicao do classificador
        #print ('Predicting...')
        y_pred = neigh.predict(X_test)

        # mostra o resultado do classificador na base de teste
        print ('Accuracy: ',  neigh.score(X_test, y_test))



        #salva no vetor de acurácias
        vet_acur.append(neigh.score(X_test, y_test))



        # cria a matriz de confusao
        cm = confusion_matrix(y_test, y_pred)

        #salva no vetor de confusoes
        vet_confusion.append(cm)


#         print (cm)
#         print(classification_report(y_test, y_pred, labels=[0,1,2,3,4,5,6,7,8,9]))
```

In [3]:

```
# archive = 'features_1.txt'

# main(archive)
```

In [3]:

```
vet_k = [1,2,3,4,5,6,7,8,9,10]
```

In [4]:

```
euclidean = []
euclidean_confusion = []

manhattan = []
manhattan_confusion = []

minkowski = []
minkowski_confusion = []

chebyshev = []
chebyshev_confusion = []

vet_acur = []
vet_confusion = []
```

In [5]:

```
method = ['euclidean', 'manhattan', 'minkowski', 'chebyshev']
```

```
PARA PEGAR O VALOR DA ACURÁCIA PARA DIFERENTES TAMANHOS QUE GERAM O VETOR DE
CARACTERÍSTICAS [X,Y]
```

In [6]:

```
arquivos = ['features_4.txt','features_2.txt','features_1.txt','features_3.txt','features_7

for archive in arquivos:
    main(archive, 3, 'chebyshev', vet_acur, vet_confusion)
```

```
Accuracy:  0.071
Accuracy:  0.131
Accuracy:  0.087
Accuracy:  0.108
Accuracy:  0.163
Accuracy:  0.096
Accuracy:  0.145
Accuracy:  0.145
Accuracy:  0.11
Accuracy:  0.145
```

```
PARA PEGAR O VALOR DA ACURÁCIA PARA DIFERENTES VALORES DE K
```

In [7]:

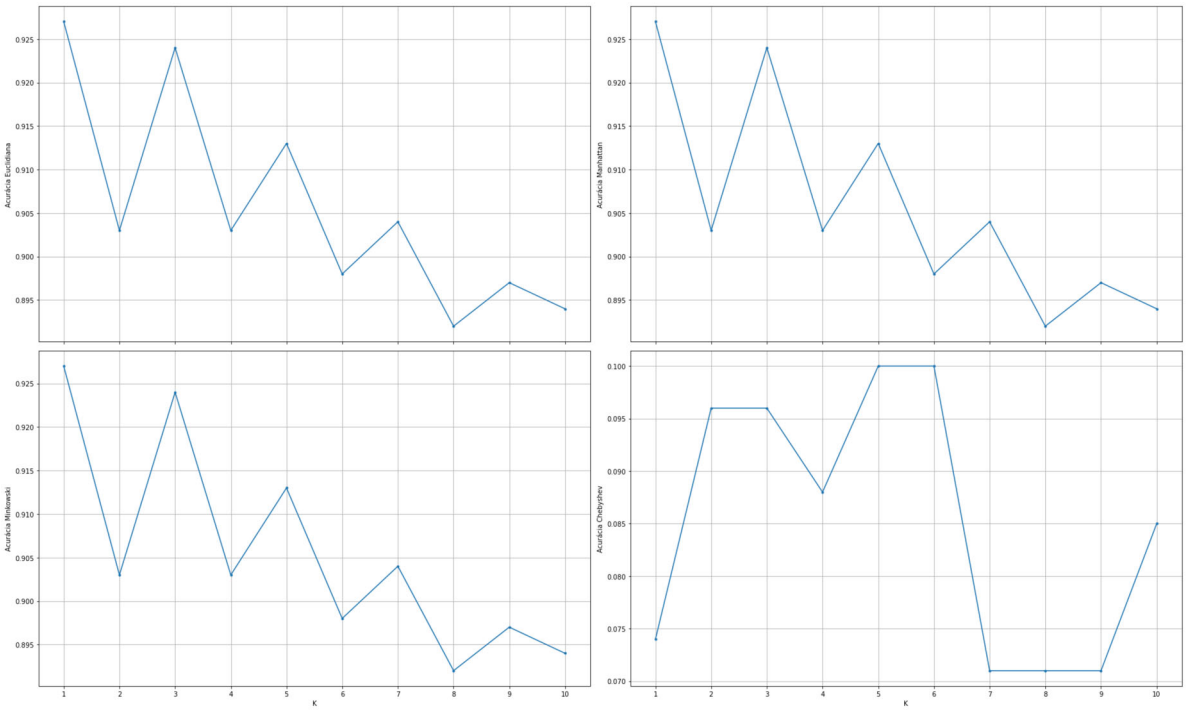
```
archive = 'features_6.txt'

for distance in method:
    vet_acur = []
    vet_confusion = []
    for k_value in range(1,11,1):
        print(distance,': k =',k_value)
        main(archive, k_value, distance, vet_acur, vet_confusion)

    if distance == 'euclidean':
        euclidean = vet_acur.copy()
        euclidean_confusion = vet_confusion.copy()
    elif distance == 'manhattan':
        manhattan = vet_acur.copy()
        manhattan_confusion = vet_confusion.copy()
    elif distance == 'minkowski':
        minkowski = vet_acur.copy()
        minkowski_confusion = vet_confusion.copy()
    else:
        chebyshev = vet_acur.copy()
        chebyshev_confusion = vet_confusion.copy()
```

```
euclidean : k = 1
Accuracy:  0.927
euclidean : k = 2
Accuracy:  0.903
euclidean : k = 3
Accuracy:  0.924
euclidean : k = 4
Accuracy:  0.903
euclidean : k = 5
Accuracy:  0.913
euclidean : k = 6
Accuracy:  0.898
euclidean : k = 7
Accuracy:  0.904
euclidean : k = 8
Accuracy:  0.892
euclidean : k = 9
Accuracy:  0.897
euclidean : k = 10
Accuracy:  0.894
manhattan : k = 1
Accuracy:  0.927
manhattan : k = 2
Accuracy:  0.903
manhattan : k = 3
Accuracy:  0.924
manhattan : k = 4
Accuracy:  0.903
manhattan : k = 5
Accuracy:  0.913
manhattan : k = 6
Accuracy:  0.898
manhattan : k = 7
Accuracy:  0.904
manhattan : k = 8
Accuracy:  0.892
manhattan : k = 9
```

```
Accuracy:  0.897
manhattan : k = 10
Accuracy:  0.894
minkowski : k = 1
Accuracy:  0.927
minkowski : k = 2
Accuracy:  0.903
minkowski : k = 3
Accuracy:  0.924
minkowski : k = 4
Accuracy:  0.903
minkowski : k = 5
Accuracy:  0.913
minkowski : k = 6
Accuracy:  0.898
minkowski : k = 7
Accuracy:  0.904
minkowski : k = 8
Accuracy:  0.892
minkowski : k = 9
Accuracy:  0.897
minkowski : k = 10
Accuracy:  0.894
chebyshev : k = 1
Accuracy:  0.074
chebyshev : k = 2
Accuracy:  0.096
chebyshev : k = 3
Accuracy:  0.096
chebyshev : k = 4
Accuracy:  0.088
chebyshev : k = 5
Accuracy:  0.1
chebyshev : k = 6
Accuracy:  0.1
chebyshev : k = 7
Accuracy:  0.071
chebyshev : k = 8
Accuracy:  0.071
chebyshev : k = 9
Accuracy:  0.071
chebyshev : k = 10
Accuracy:  0.085
```

In [11]:

```
fig, ([ax1, ax2], [ax3, ax4]) = plt.subplots(2,2,figsize=(25,15), sharex=True)
#fig.suptitle('Comparativo da acurácia entre o uso de diferentes métricas de distâncias par

ax1.plot(vet_k, euclidean,'.-')
# ax1.set_xticklabels(x_labels, rotation=45, ha='right')
ax1.set_ylabel('Acurácia Euclidiana')
ax1.grid()


ax2.plot(vet_k, manhattan, '.-')
# ax2.set_xticklabels(x_labels, rotation=45, ha='right')
ax2.set_ylabel('Acurácia Manhattan')
ax2.grid()


ax3.plot(vet_k, minkowski, '.-')
# ax3.set_xticklabels(x_labels, rotation=45, ha='right')
ax3.set_ylabel('Acurácia Minkowski')
ax3.set_xlabel('K')
ax3.grid()


ax4.plot(vet_k, chebyshev, '.-')
# ax4.set_xticklabels(x_labels, rotation=45, ha='right')
ax4.set_ylabel('Acurácia Chebyshev')
ax4.set_xlabel('K')
ax4.grid()


plt.xticks(vet_k)
fig.tight_layout()
plt.show()
# fig.savefig('k_variation.png', dpi=fig.dpi)
```

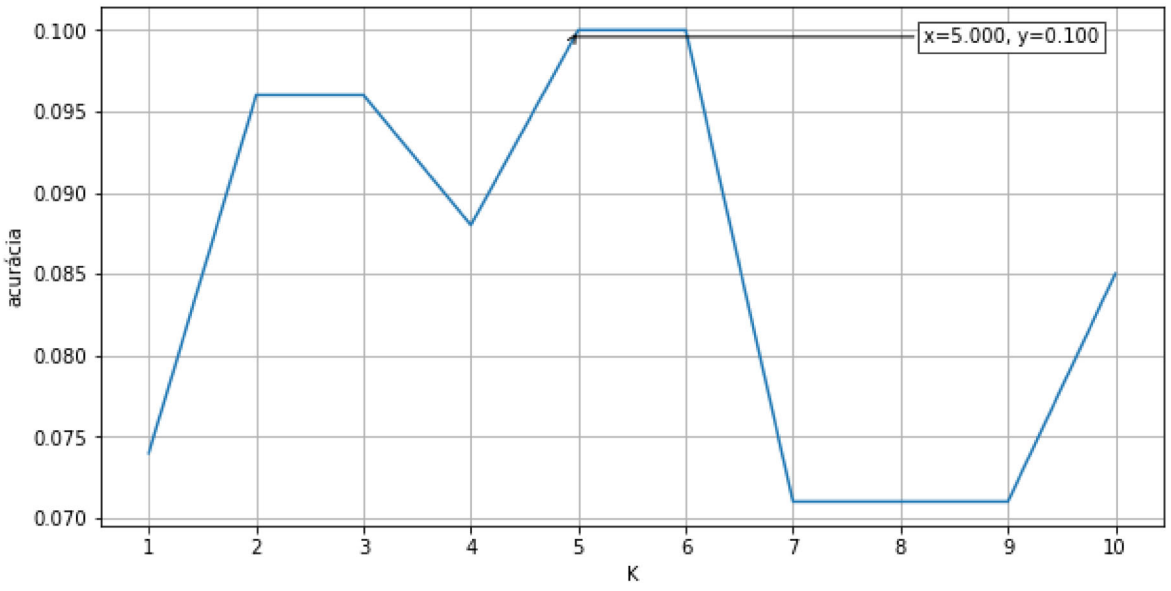


In [ ]:

```
#https://stackoverflow.com/questions/43374920/how-to-automatically-annotate-maximum-value-i
#https://matplotlib.org/stable/gallery/subplots_axes_and_figures/subplots_demo.html
```

In [17]:

```
fig, ax = plt.subplots(figsize=(10, 5))

def annot_max(x,y, ax=None):
    xmax = x[np.argmax(y)]
    ymax = y.max()
    text= "x={:.3f}, y={:.3f}".format(xmax, ymax)
    if not ax:
        ax=plt.gca()
    bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
    arrowprops=dict(arrowstyle="->",connectionstyle="angle,angleA=0,angleB=60")
    kw = dict(xycoords='data',textcoords="axes fraction",
              arrowprops=arrowprops, bbox=bbox_props, ha="right", va="top")
    ax.annotate(text, xy=(xmax, ymax), xytext=(0.94,0.96), **kw)

ax.plot(vet_k, chebyshev)


tempx = np.array(vet_k)
tempy = np.array(chebyshev)
annot_max(tempx,tempy)

ax.set(xlabel='K', ylabel='acurácia')
# ax.set_xticklabels(x_labels, rotation=45, ha='right')
ax.grid()

plt.xticks(vet_k)

fig.savefig("chebyshev.png")
plt.show()
```



In [28]:

```
print('-----MATRIZ DE CONFUSÃO (DISTÂNCIA EUCLIDIANA) -----')
print('- K=1\n', euclidean_confusion[0])
print('\n\n')
print('- K=10\n', euclidean_confusion[9])
```

```
-----MATRIZ DE CONFUSÃO (DISTÂNCIA EUCLIDIANA) -----
- K=1
 [[ 94   1   0   0   0   1   1   0   0   0]
 [  0  93   0   0   0   1   0   0   0   1]
 [  1   1 102   1   0   0   0   4   1   1]
 [  0   1   0  98   0   1   0   0   3   0]
 [  0   8   0   0  83   1   1   0   0   2]
 [  1   0   0   6   0  89   0   0   1   0]
 [  1   5   0   0   0   0 100   0   0   0]
 [  0   3   1   0   1   0   0  88   0   4]
 [  0   3   0   1   1   2   0   1  79   0]
 [  0   1   0   0   1   0   0   9   0 101]]



- K=10
 [[95  1  0  0  0  1  0  0  0  0]
 [ 0 94  0  1  0  0  0  0  0  0]
 [ 2  6 96  0  0  0  2  4  0  1]
 [ 0  1  1 98  0  0  0  1  2  0]
 [ 0 12  0  0 81  0  1  0  0  1]
 [ 1  0  0  7  0 88  1  0  0  0]
 [ 2  7  0  0  0  0 97  0  0  0]
 [ 0 10  0  0  1  0  0 82  0  4]
 [ 0  7  0  2  0  1  0  5 71  1]
 [ 0  3  0  0  4  0  0 13  0 92]]
```

In [29]:

```
print('-----MATRIZ DE CONFUSÃO (DISTÂNCIA DE MANHATTAN) -----')
print('- K=1\n', manhattan_confusion[0])
print('\n\n')
print('- K=10\n', manhattan_confusion[9])
```

```
-----MATRIZ DE CONFUSÃO (DISTÂNCIA DE MANHATTAN) -----
- K=1
 [[ 94   1   0   0   0   1   1   0   0   0]
 [  0  93   0   0   0   1   0   0   0   1]
 [  1   1 102   1   0   0   0   4   1   1]
 [  0   1   0  98   0   1   0   0   3   0]
 [  0   8   0   0  83   1   1   0   0   2]
 [  1   0   0   6   0  89   0   0   1   0]
 [  1   5   0   0   0   0 100   0   0   0]
 [  0   3   1   0   1   0   0  88   0   4]
 [  0   3   0   1   1   2   0   1  79   0]
 [  0   1   0   0   1   0   0   9   0 101]]



- K=10
 [[95  1  0  0  0  1  0  0  0  0]
 [ 0 94  0  1  0  0  0  0  0  0]
 [ 2  6 96  0  0  0  2  4  0  1]
 [ 0  1  1 98  0  0  0  1  2  0]
 [ 0 12  0  0 81  0  1  0  0  1]
 [ 1  0  0  7  0 88  1  0  0  0]
 [ 2  7  0  0  0  0 97  0  0  0]
 [ 0 10  0  0  1  0  0 82  0  4]
 [ 0  7  0  2  0  1  0  5 71  1]
 [ 0  3  0  0  4  0  0 13  0 92]]
```

In [30]:

```
print('-----MATRIZ DE CONFUSÃO (DISTÂNCIA DE MINKOWSKI) -----')
print('- K=1\n', minkowski_confusion[0])
print('\n\n')
print('- K=10\n', minkowski_confusion[9])
```

```
-----MATRIZ DE CONFUSÃO (DISTÂNCIA DE MINKOWSKI) -----
- K=1
 [[ 94   1   0   0   0   1   1   0   0   0]
 [  0  93   0   0   0   1   0   0   0   1]
 [  1   1 102   1   0   0   0   4   1   1]
 [  0   1   0  98   0   1   0   0   3   0]
 [  0   8   0   0  83   1   1   0   0   2]
 [  1   0   0   6   0  89   0   0   1   0]
 [  1   5   0   0   0   0 100   0   0   0]
 [  0   3   1   0   1   0   0  88   0   4]
 [  0   3   0   1   1   2   0   1  79   0]
 [  0   1   0   0   1   0   0   9   0 101]]



- K=10
 [[95  1  0  0  0  1  0  0  0  0]
 [ 0 94  0  1  0  0  0  0  0  0]
 [ 2  6 96  0  0  0  2  4  0  1]
 [ 0  1  1 98  0  0  0  1  2  0]
 [ 0 12  0  0 81  0  1  0  0  1]
 [ 1  0  0  7  0 88  1  0  0  0]
 [ 2  7  0  0  0  0 97  0  0  0]
 [ 0 10  0  0  1  0  0 82  0  4]
 [ 0  7  0  2  0  1  0  5 71  1]
 [ 0  3  0  0  4  0  0 13  0 92]]
```

In [31]:

```
print('-----MATRIZ DE CONFUSÃO (DISTÂNCIA DE MINKOWSKI) -----')
print('- K=5\n', chebyshev_confusion[4])
print('\n\n')
print('- K=7\n', chebyshev_confusion[6])
```

```
-----MATRIZ DE CONFUSÃO (DISTÂNCIA DE MINKOWSKI) -----
- K=5
 [[ 0  0 96  0  1  0  0  0  0  0]
 [ 0  0 87  0  8  0  0  0  0  0]
 [ 1  0 63  0 27  0  0 20  0  0]
 [ 0  0 83  0  5  0  0 15  0  0]
 [ 1  0 86  0  5  0  0  3  0  0]
 [ 0  0 89  0  3  0  0  5  0  0]
 [ 0  0 91  0  9  1  0  5  0  0]
 [ 2  0 56  0  6  1  0 32  0  0]
 [ 0  0 72  0  2  0  0 13  0  0]
 [ 0  0 93  0  2  0  0 17  0  0]]



- K=7
 [[ 0  0 96  0  1  0  0  0  0  0]
 [ 0  0 86  0  8  0  0  0  0  1]
 [ 1  0 63  0 47  0  0  0  0  0]
 [ 0  0 81  0 20  0  0  0  0  2]
 [ 1  0 86  0  8  0  0  0  0  0]
 [ 0  0 89  0  8  0  0  0  0  0]
 [ 0  0 91  0 14  1  0  0  0  0]
 [ 2  0 54  0 38  1  0  0  0  2]
 [ 0  0 72  0 15  0  0  0  0  0]
 [ 0  0 93  0 19  0  0  0  0  0]]
```