Main ideas for the final solution (not a complete solution):

```
def distance(x):
return np.linalg.norm(x-p)
```

FOR K=2

```
# First, we compute tuples of (distance to p, label)
distances_rdd = data_rdd.map(lambda inst: (distance(inst.features), inst.label))
# Then, we compute the minimum distance
min_distance = distances_rdd.map(lambda (dist, label): dist).reduce(min)
# Then, we filter the tuple with exactly that distance and get the label
class1 = distances_rdd.filter(lambda (dist, label): dist==min_distance).take(1)[0][1]

# In order to get the second closest instance, we remove the instance with the closest distance:
distances2_rdd = distances_rdd.filter(lambda (dist, label): dist>min_dist)
# and would compute class2 as we did with class1 (not done here).
```

Note 1: everytime an action is performed on distances_rdd, it is recomputed. Therefore, a more efficient solution would force distances_rdd to persist the first time it is defined, like this:

```
distances_rdd = data_rdd.map(lambda inst: (inst, distance(inst.features)))
distances_rdd.persist()
```

Note 2: the former ideas assume that there are no two closest instances with the same distance, which is true for this particular point p. This could be checked with:

```
distances_rdd.filter(lambda (dist, label): dist==min_distance).collect()
```