Diagnosing diabetes with KNN

In this small notebook we use data about diabetes in female Pima Indians to see how well we can predict diabetes given variables like age, glucose level, blood pressure, and skin thickness.

We focus on two things:

- data exploration and cleaning
- using hyperparameter tuning to find the best hyperparameter values for KNN classification

The dataset was downloaded from Kaggle on October 21, 2021.

https://www.kaggle.com/uciml/pima-indians-diabetes-database (https://www.kaggle.com/uciml/pima-indians-diabetes-database)

Information about the data set can be found on the Kaggle page.

Instructions

- Please read the entire notebook carefully!
- Note that plots are preceded by a question and followed by interpretation of the plot.
- Each problem cell begins with #@.
- Do not make changes outside the problem cells.
- Be sure to include plot titles, labels, etc. as shown in model output.
- Run your code from top to bottom before submitting.
- Do not modify the file name.

Out[2]: Click here to display/hide the code.

Set the random seed for repeatability

Read the data

It is useful to identify the predictor and target variables right away.

Data exploration

Looking at an overview of the data, we see no NA values. All variables are numeric.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

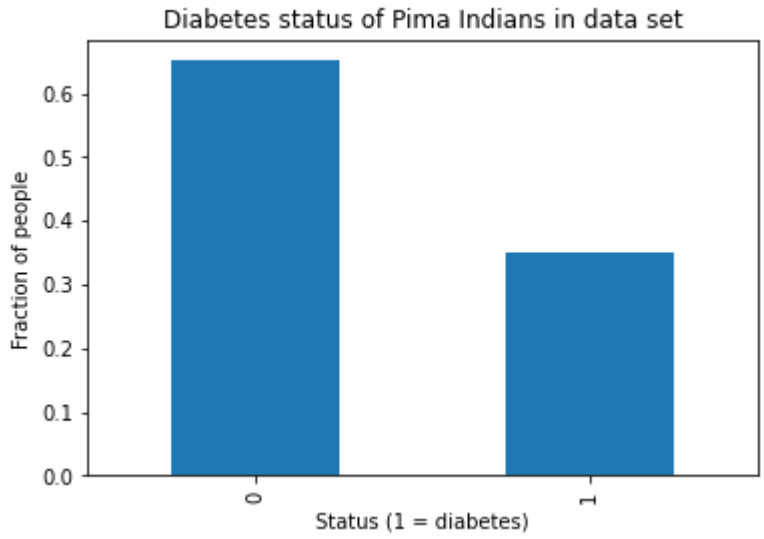
It is helpful to look at a little of the raw data.

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
0	6	148	72	35	0	33.6	0.627	
1	1	85	66	29	0	26.6	0.351	
2	8	183	64	0	0	23.3	0.672	
3	1	89	66	23	94	28.1	0.167	
4	0	137	40	35	168	43.1	2.288	

'Outcome' is the target value. A value of 1 indicates the presence of diabetes.

How many people represented in the data have diabetes?



About 1/3 of the patients have diabetes, according to the data.

What are the basic statistics of the numeric variables?

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
count	768.00	768.00	768.00	768.00	768.00	768.00	76
mean	3.85	120.89	69.11	20.54	79.80	31.99	
std	3.37	31.97	19.36	15.95	115.24	7.88	
min	0.00	0.00	0.00	0.00	0.00	0.00	
25%	1.00	99.00	62.00	0.00	0.00	27.30	
50%	3.00	117.00	72.00	23.00	30.50	32.00	
75%	6.00	140.25	80.00	32.00	127.25	36.60	
max	17.00	199.00	122.00	99.00	846.00	67.10	

We see that there are no negative values in the dataset.

Something that looks odd is that many variables have zero as their minimum value. Can blood pressure really be zero? What about skin thickness, and body mass index? Perhaps some of the zeroes indicate bad data.

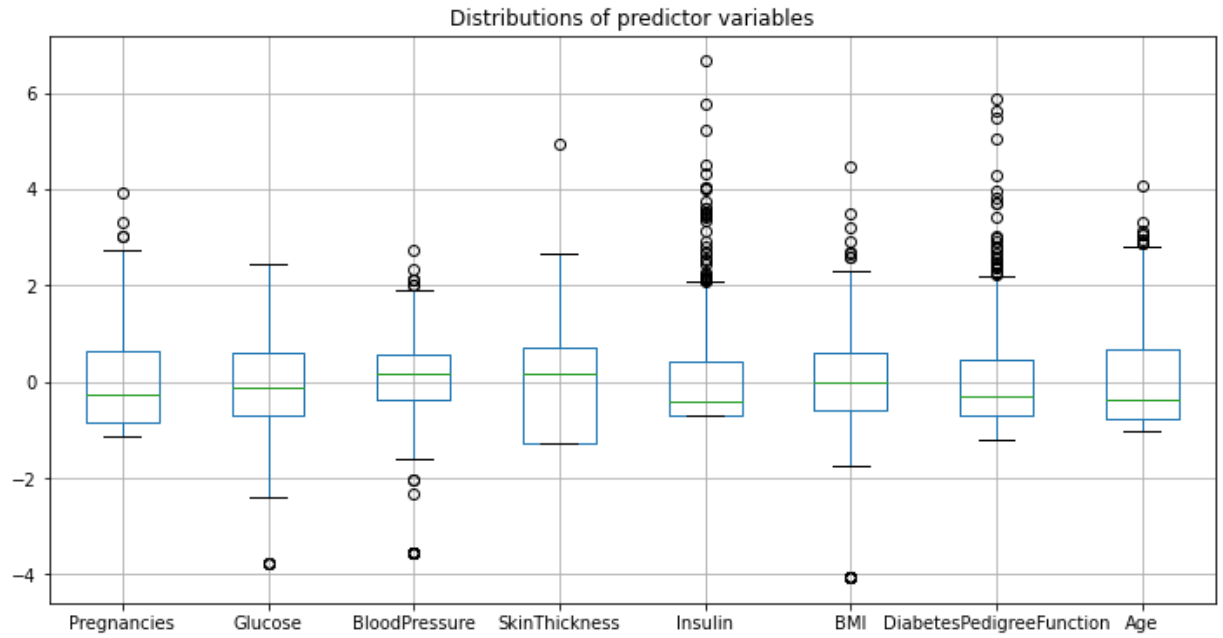
Does the data contain outliers? This will be easier to see if the data is scaled.

Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
count	768.00	768.00	768.00	768.00	768.00	768.00	76
mean	0.00	0.00	-0.00	0.00	-0.00	0.00	
std	1.00	1.00	1.00	1.00	1.00	1.00	
min	-1.14	-3.78	-3.57	-1.29	-0.69	-4.06	-
25%	-0.84	-0.69	-0.37	-1.29	-0.69	-0.60	-
50%	-0.25	-0.12	0.15	0.15	-0.43	0.00	-
75%	0.64	0.61	0.56	0.72	0.41	0.58	
max	3.91	2.44	2.73	4.92	6.65	4.46	

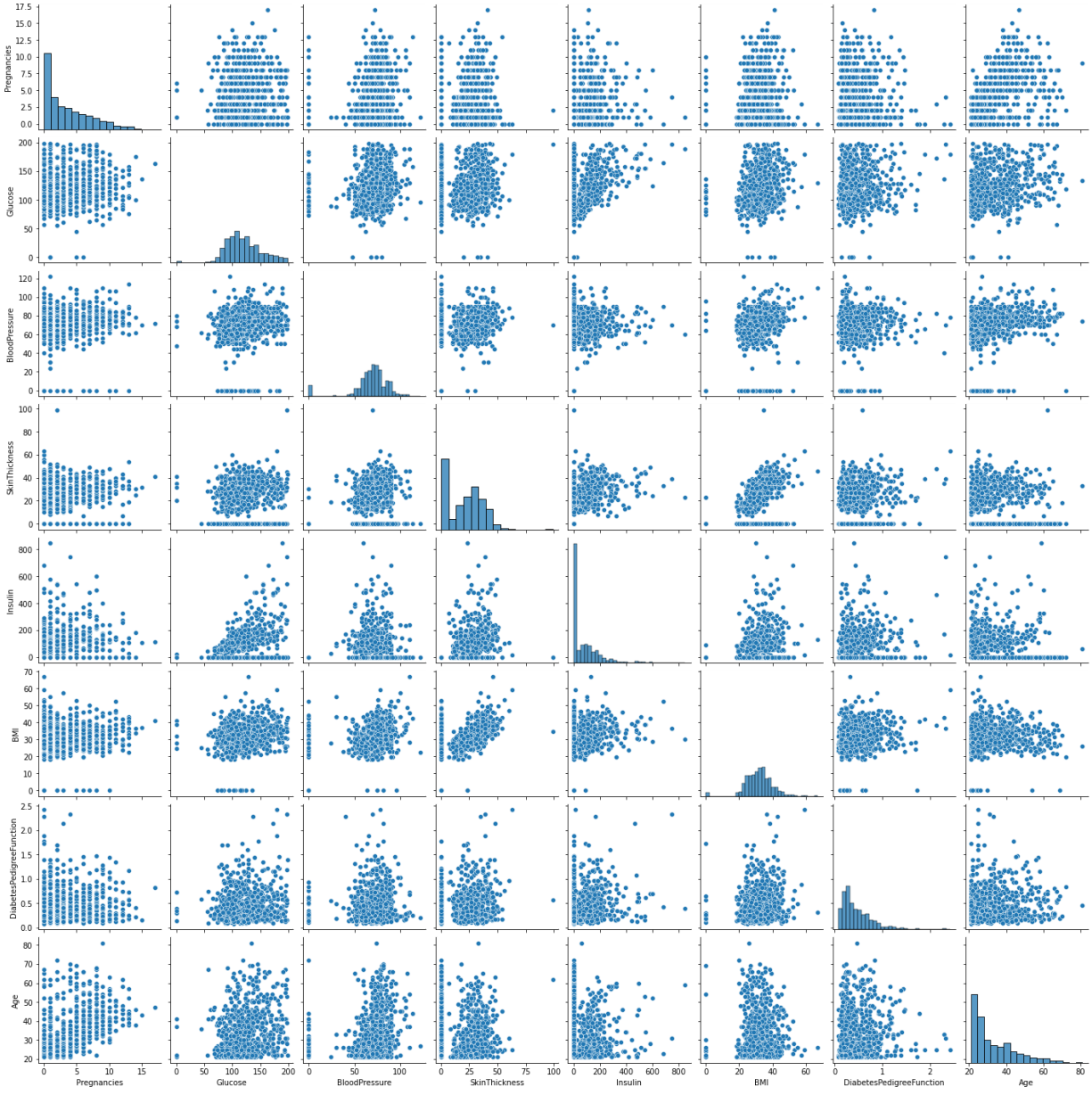
We see that the maximum insulin value is more than 6 standard deviations above the mean, and the max skin thickness and diabetes pedigree function values are also large.

Box plots can help identify outliers. When showing multiple boxplots at once, scaling is useful.



There seem to be many outliers in the Insulin and DiabetesPedegreeFunction variables.

We can get a deeper feeling for zero values and outliers by plotting the data.

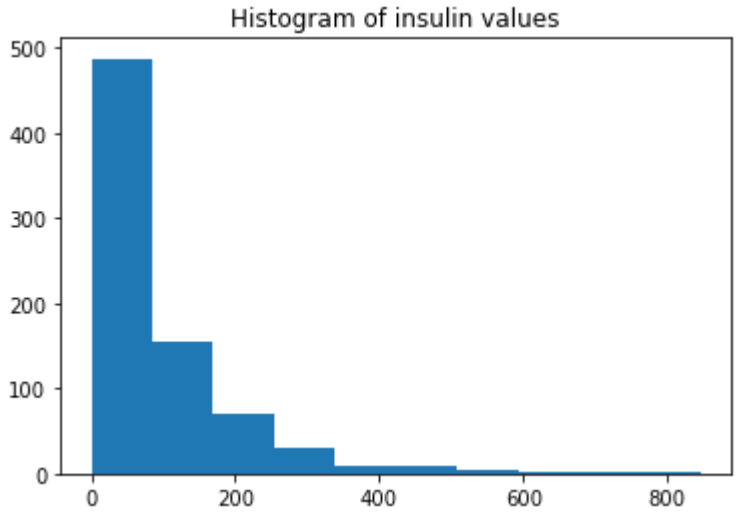

Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age

We find that some of the 0 values do indeed look strange. For example, the zero BMI values look strange. Also, some of the max values look like outliers. For example, the largest skin thickness value.

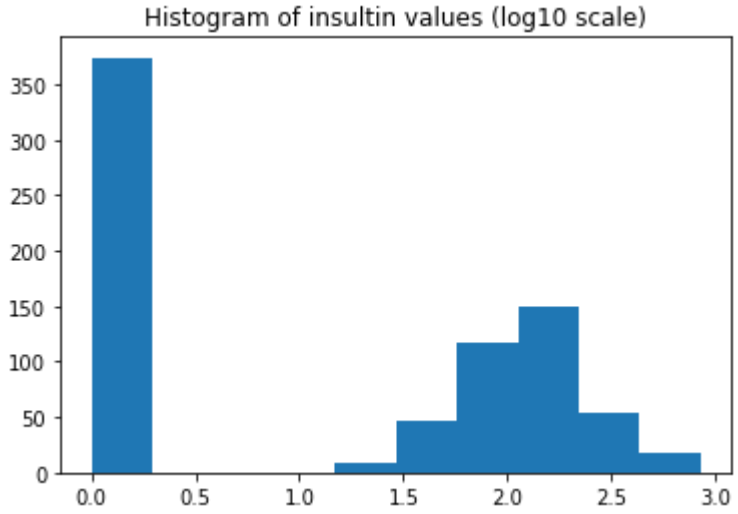
Investigating zero values

A concern is that zero values might represent missing data.

Let's focus first on insulin. What is the distribution of insulin values?



The distribution is highly skewed. Plotting the log may make the distribution clearer.



This picture makes the zero insulin values look very suspicious.

What about the other predictors? For each predictor, what fraction of the values are 0?

Out[15]:
```
Insulin                     0.487
SkinThickness               0.296
Pregnancies                 0.145
BloodPressure               0.046
BMI                         0.014
Glucose                     0.007
DiabetesPedigreeFunction    0.000
Age                         0.000
dtype: float64
```

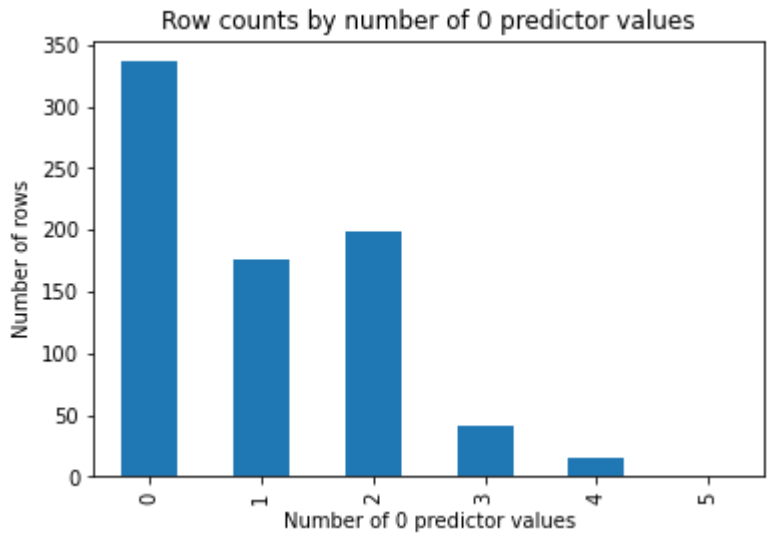
About 46% of the insulin values are zero. This could suggest they are valid. Also, a little background research suggests zero insulin values are possible.

Another way to understand zero values is to see how far they are away from the mean value.

Out[16]:
```
BMI                         4.058
Glucose                     3.781
BloodPressure               3.570
Age                         2.827
DiabetesPedigreeFunction    1.424
SkinThickness               1.287
Pregnancies                 1.141
Insulin                     0.692
dtype: float64
```

This backs up the idea that blood pressure values of 0 represent missing data.

Are the 0 values in the data set clustered in some rows? In other words, are the 0's spread evenly across people, or clustered in some people? To look into this, we can count the number of rows with no zero values, with 1 zero value, etc.



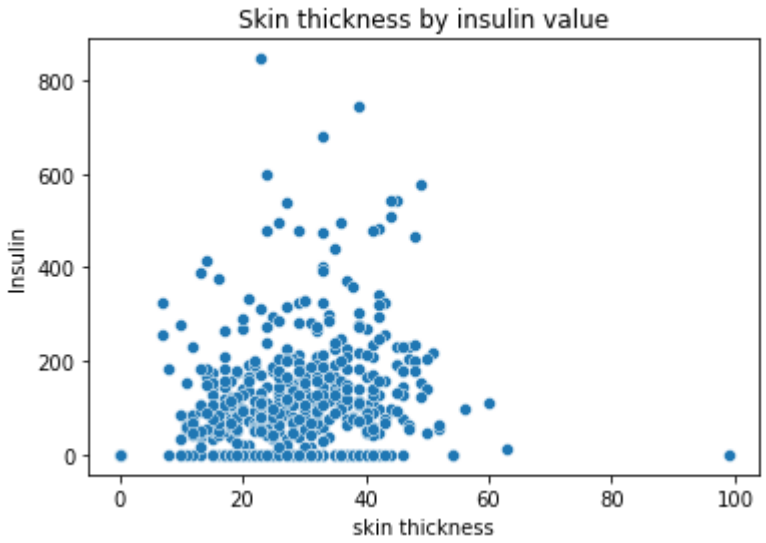
We see that about 170 rows contain two zero values, but very few rows contain more than two zero values.

In the rows with more than one zero value, which predictors are 0?

Out[18]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
2	8	183	64	0	0	23.3	0.672
5	5	116	74	0	0	25.6	0.201
7	10	115	0	0	0	35.3	0.134
9	8	125	96	0	0	0.0	0.232
10	4	110	92	0	0	37.6	0.191
11	10	168	74	0	0	38.0	0.537
12	10	139	80	0	0	27.1	1.441
15	7	100	0	0	0	30.0	0.484
17	7	107	74	0	0	29.6	0.254
21	8	99	84	0	0	35.4	0.388

Zero values for skin thickness and insulin seem to go together. Does a scatter plot confirm this idea?



It seems that if skin thickness is 0, then insulin is 0, because there are no skin thickness values of 0 except where insulin is 0.

Perhaps if a person's skin is very thin, it is hard to test for insulin. Talking to a diabetes specialist would help in understanding this.

Data preprocessing

Our strategy on zero values will be to remove rows in which BMI, Glucose, BloodPressure, or SkinThickness are 0. An alternative approach would be to impute values for these zero values.

Use describe again to see the result of removing these rows.

Out[21]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
count	532.000000	532.000000	532.000000	532.000000	532.000000	532.000000	
mean	3.516917	121.030075	71.505639	29.182331	114.988722	32.890226	
std	3.312036	30.999226	12.310253	10.523878	123.007555	6.881109	
min	0.000000	56.000000	24.000000	7.000000	0.000000	18.200000	
25%	1.000000	98.750000	64.000000	22.000000	0.000000	27.875000	
50%	2.000000	115.000000	72.000000	29.000000	91.500000	32.800000	
75%	5.000000	141.250000	80.000000	36.000000	165.250000	36.900000	
max	17.000000	199.000000	110.000000	99.000000	846.000000	67.100000	

Put the predictor and target values into NumPy arrays.

```
X shape: (532, 8)
y shape: (532,)
```

Perform an 75/25 test/train split.

Scale the data using z-score normalization. Note that the scaler is trained on the training data, and the trained scaler is used on both the training and test data. The target values are not scaled.

Basic KNN classification

Out[26]:

```
KNeighborsClassifier()
```

Three important hyperparameters of a KNN classifier:

- the number of nearest neighbors ('n_neighbors' in KNeighborsClassifier())
- the distance metric ('metric' and 'p')
- whether closer neighbors should be more important when making predictions ('weights')

If 'metric' is 'minkowski', then p = 1 means manhattan distance, and p = 2 means Euclidian distance.

Let's look at the default hyperparameters for the KNN classifier.

Out[27]:
```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'uniform'}
```

What test accuracy is achieved with the default KNN classifier?

Out[29]:
```
0.754423076923077
```

```
CV accuracy using default hyperparameters: 0.754
```

Compute the accuracy we'd get by always predicting that "no diabetes".

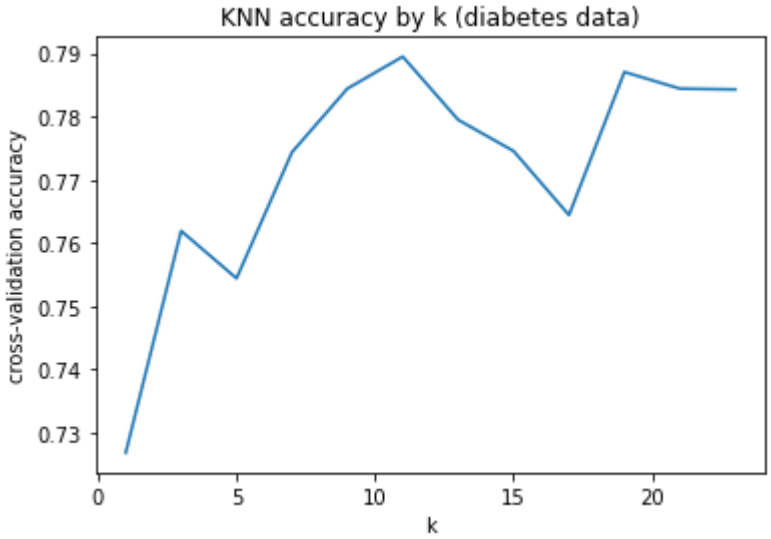
When we compute the accuracy of a classifier, we want a baseline for comparison. The usual baseline is the accuracy that you would get if you always predicted the most common value of the predictor variable. In this case, the most common value of y_test is 0.

```
Baseline accuracy: 0.684
```

So the test accuracy is significantly better than the baseline.

Determine best k by using 10-fold cross validation

The default value of k is 5 with KNeighborsClassifier. Is this a good value for k?



The plot shows that a value of 11 is best, but 19 is almost as good. Perhaps any value between about 11 and 21 is good.

Find best combination of hyperparameters k and p using grid search

The problem with finding the best k value on its own, is that the best value of k might depend on the other hyperparameter values, such as distance function.

We really would like to search for the best combination of parameter values. Grid search is a good way to do this.

Out[35]:
```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': range(5, 16, 2), 'p': [1, 2],
                         'weights': ['uniform', 'distance']})
```

```
{'n_neighbors': 11, 'p': 2, 'weights': 'distance'}
```

```
best CV accuracy: 0.789
```

The CV accuracy we get with the best hyperparameter values is better than with the default hyperparameters.

Train model with best hyperparameters on all training data.

Now that we've tuned our classifier, we will train it using all training data.

Out[38]:
```
KNeighborsClassifier(n_neighbors=11, weights='distance')
```

Compute test accuracy using the score() function

We have not used the test data yet. We now compute test accuracy to see how our classifier does on data never seen before.

```
Test accuracy: 0.759
```

Our test accuracy is significantly better than our baseline accuracy, but only by about 10%.