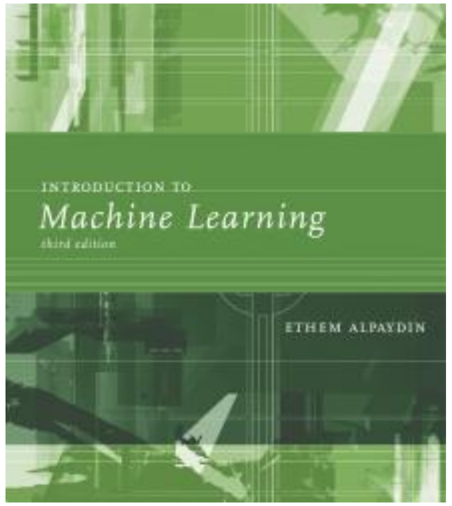


Nonparametric Methods

Lecture notes by Ethem Alpaydın
Introduction to Machine Learning (Boğaziçi Üniversitesi)

Lecture notes by Kevyn Collins-Thompson
Applied Machine Learning (Coursera)

Nonparametric vs Parametric Methods

- In parametric methods, whether for density estimation, classification, or regression, we assume a model valid over the whole input space
- The advantage of a parametric method is that it reduces the problem of estimating a probability density function, discriminant, or regression function to estimating the values of a small number of parameters.
- Its disadvantage is that this assumption does not always hold and we may incur a large error if it does not.

Nonparametric vs Parametric Methods

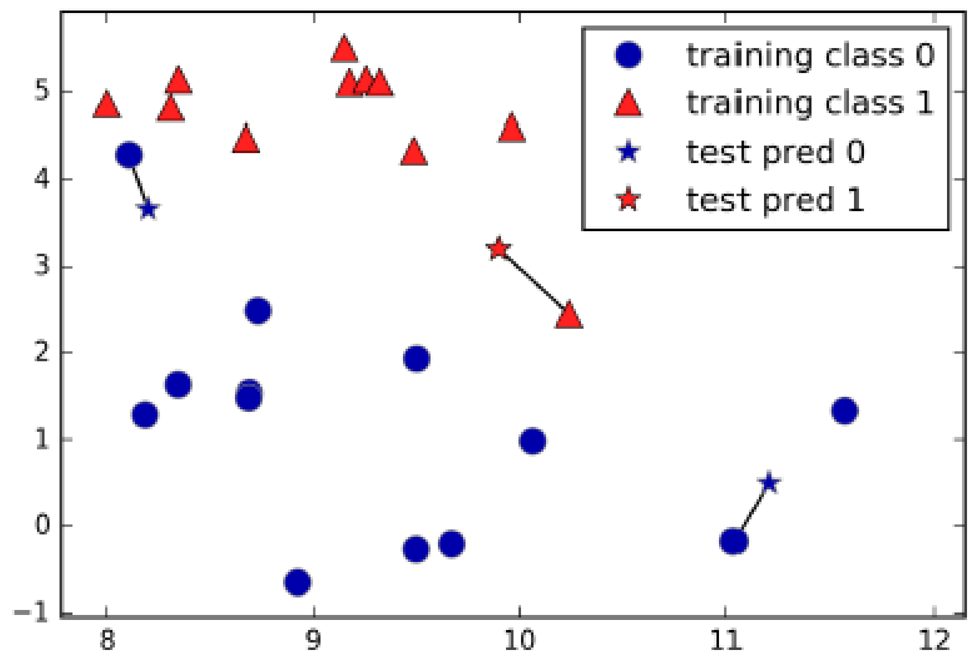
- In nonparametric estimation, all we assume is that similar inputs have similar outputs.
- Nonparametric method is composed of finding the similar past instances from the training set using a suitable distance measure and interpolating from them to find the right output.
- Aka lazy/memory-based/case-based/instance-based learning
- What they do is to store the training instances in a lookup table and interpolate from these.

k-NN classifier

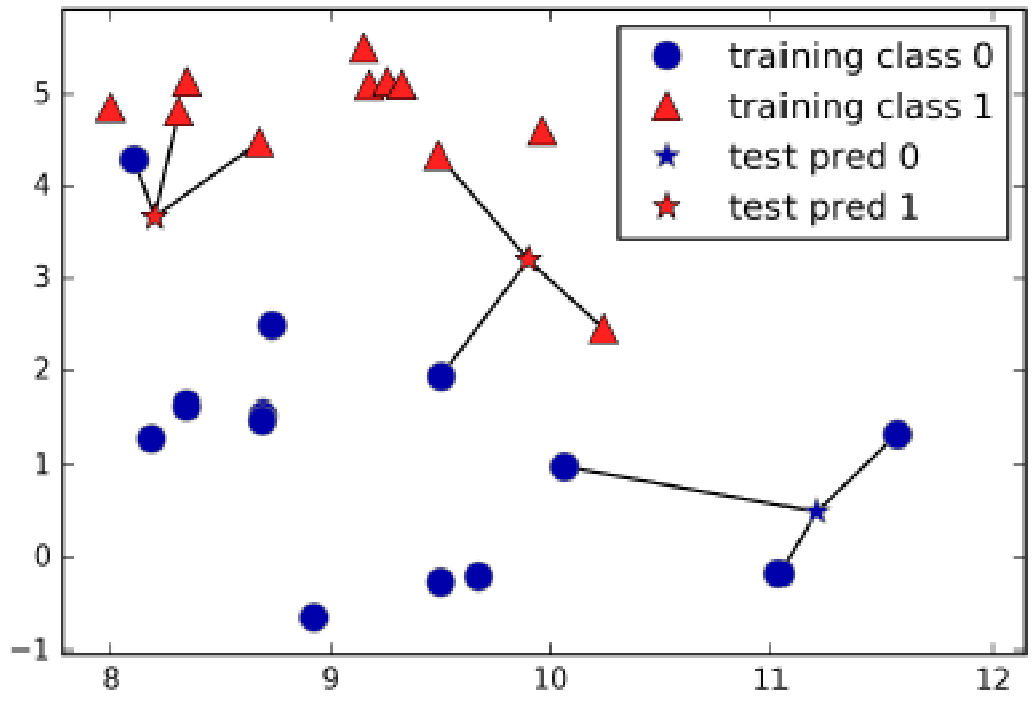
- The *k-nn classifier* assigns the input to the class having most examples among the *k* neighbors of the input.
- All neighbors have equal vote, and the class having the maximum number of voters among the *k* neighbors is chosen.
- Ties are broken arbitrarily or a weighted vote is taken.

k-NN classifier

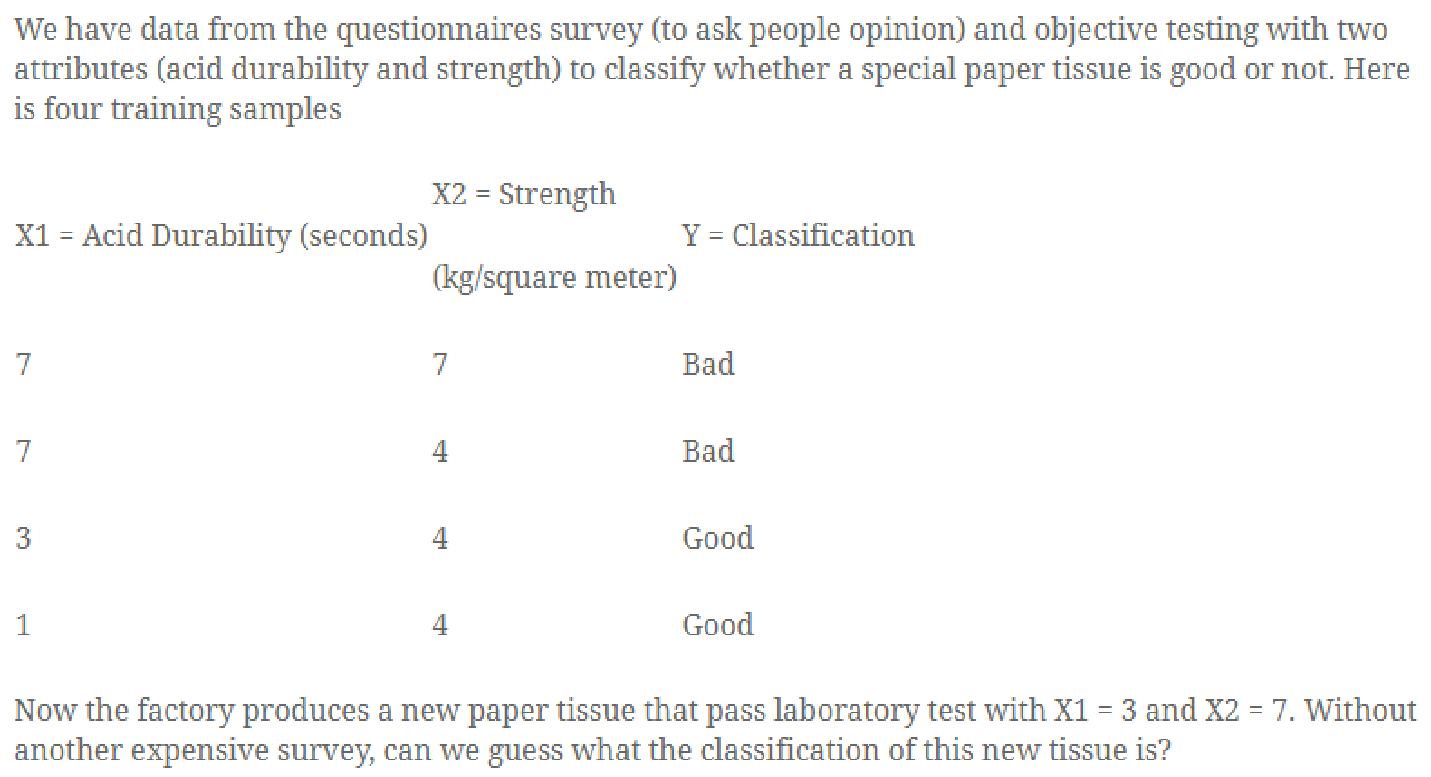
```
mglearn.plots.plot_knn_classification(n_neighbors=1)
```



```
mglearn.plots.plot_knn_classification(n_neighbors=3)
```



Example



We have data from the questionnaires survey (to ask people opinion) and objective testing with two attributes (acid durability and strength) to classify whether a special paper tissue is good or not. Here is four training samples

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Y = Classification
7	7	Bad
7	4	Bad
3	4	Good
1	4	Good

Now the factory produces a new paper tissue that pass laboratory test with X1 = 3 and X2 = 7. Without another expensive survey, can we guess what the classification of this new tissue is?

https://people.revoledu.com/kardi/tutorial/KNN/KNN_Numerical-example.html

Example

1. Determine parameter k= # of nearest neighbors

Suppose use k=3

https://people.revoledu.com/kardi/tutorial/KNN/KNN_Numerical-example.html

Example

2. Calculate the distance between the query instance (x1=3, x2=7) and all the tranining example

X1	X2	Square distance
7	7	$(7-3)^2 + (7-7)^2 = 16$
7	4	$(7-3)^2 + (4-7)^2 = 25$
3	4	$(3-3)^2 + (4-7)^2 = 9$
1	4	$(1-3)^2 + (4-7)^2 = 13$

https://people.revoledu.com/kardi/tutorial/KNN/KNN_Numerical-example.html

Example

3. Sort the distance and determine nearst neighbors based on the k-th minimum distance

X1	X2	Square distance	rank	Is it included in 3-nn?
7	7	16	3	Yes
7	4	25	4	No
3	4	9	1	Yes
1	4	13	2	Yes

https://people.revoledu.com/kardi/tutorial/KNN/KNN_Numerical-example.html

Example

4. Gather the category y of the neares neighbor

X1	X2	Square distance	rank	Is it included in 3-nn?	Category of nearest neighbor
7	7	16	3	Yes	Bad
7	4	25	4		-
3	4	9	1	Yes	Good
1	4	13	2	Yes	Good

https://people.revoledu.com/kardi/tutorial/KNN/KNN_Numerical-example.html

Example

5. Use simply majority of the category of nearest neighbors as the prediction value of the query instance

We have 2 good and 1 bad

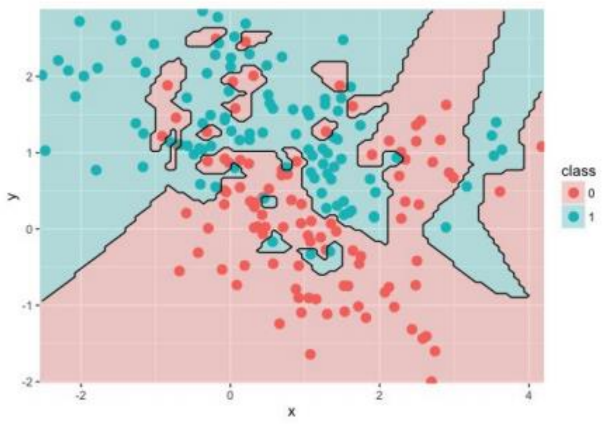
(3,7) is included in GOOD category.

https://people.revoledu.com/kardi/tutorial/KNN/KNN_Numerical-example.html

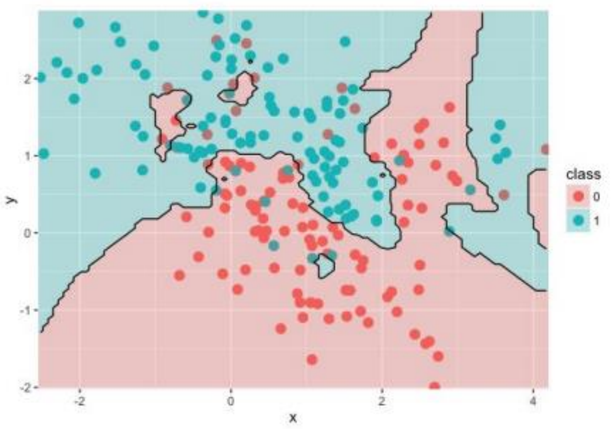
Scikit-learn

```
from sklearn.model_selection import train_test_split
X, y = mglearn.datasets.make_forge()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)

clf.fit(X_train, y_train)

print("Test set predictions: {}".format(clf.predict(X_test)))
print("Test set accuracy: {:.2f}".format(clf.score(X_test, y_test)))
```

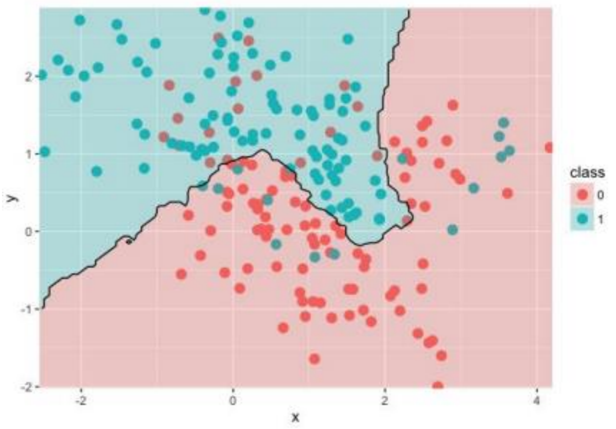
Choose k?



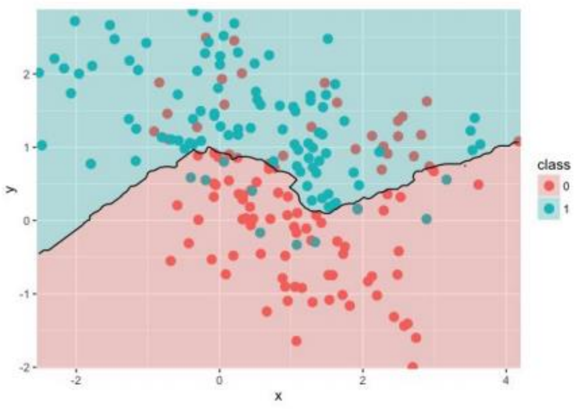
k=1



k=3



k=15

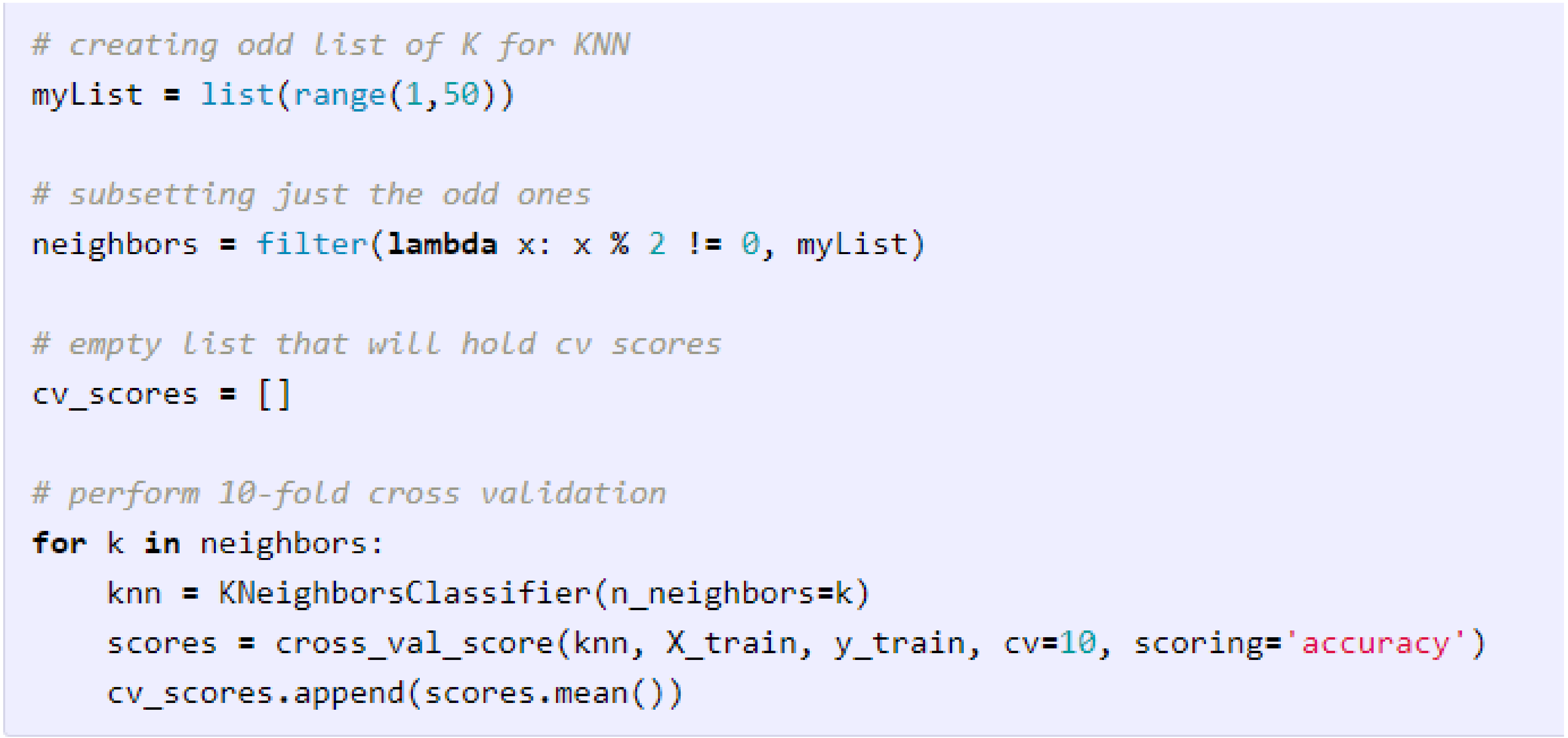


k=51

Small k: overfit
Large k: underfit

Parameter tuning with cross validation

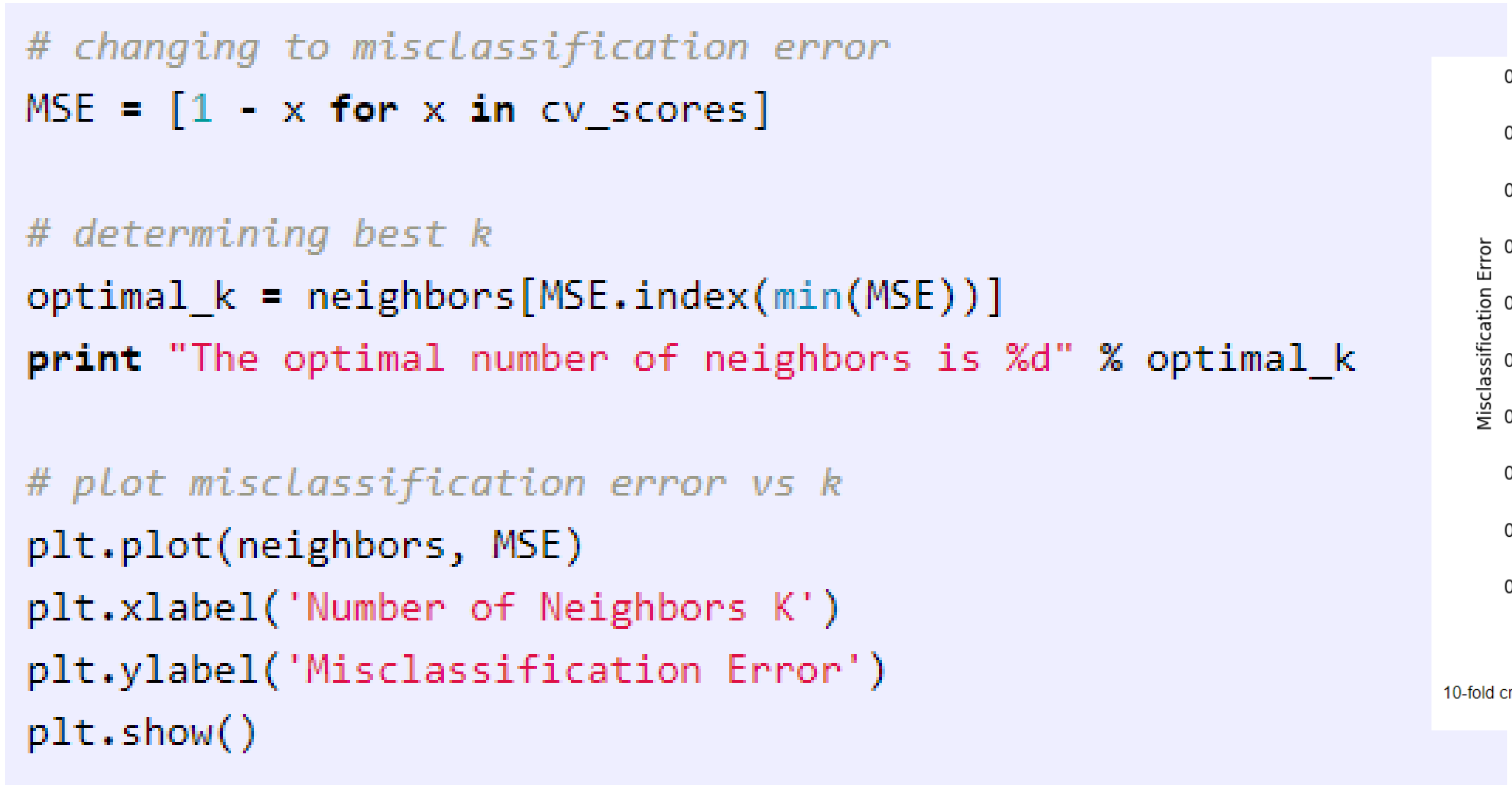
Parameter Tuning with Cross Validation



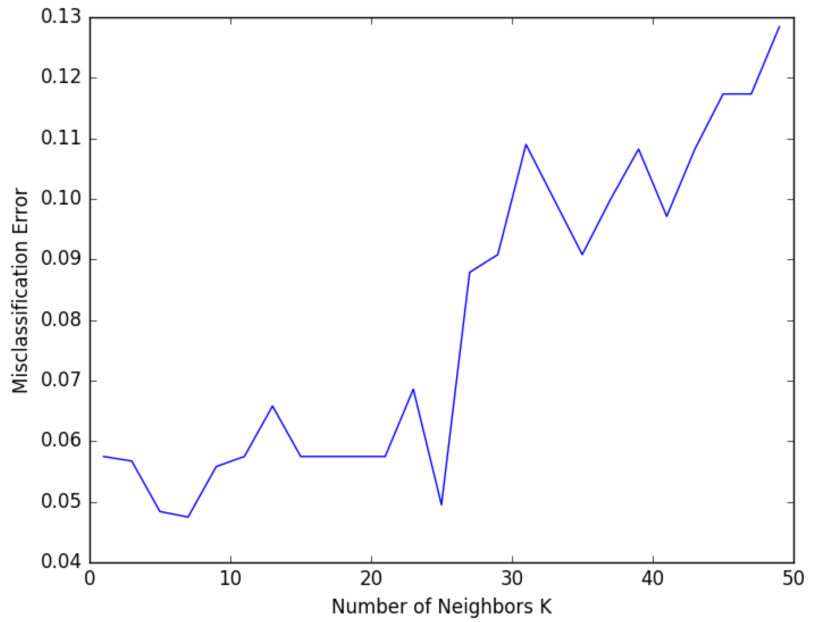
```
# creating odd list of K for KNN
myList = list(range(1,50))

# subsetting just the odd ones
neighbors = filter(lambda x: x % 2 != 0, myList)

# empty list that will hold cv scores
cv_scores = []

# perform 10-fold cross validation
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())
```

https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/

Parameter Tuning with Cross Validation



```
# changing to misclassification error
MSE = [1 - x for x in cv_scores]

# determining best k
optimal_k = neighbors[MSE.index(min(MSE))]
print "The optimal number of neighbors is %d" % optimal_k

# plot misclassification error vs k
plt.plot(neighbors, MSE)
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()
```

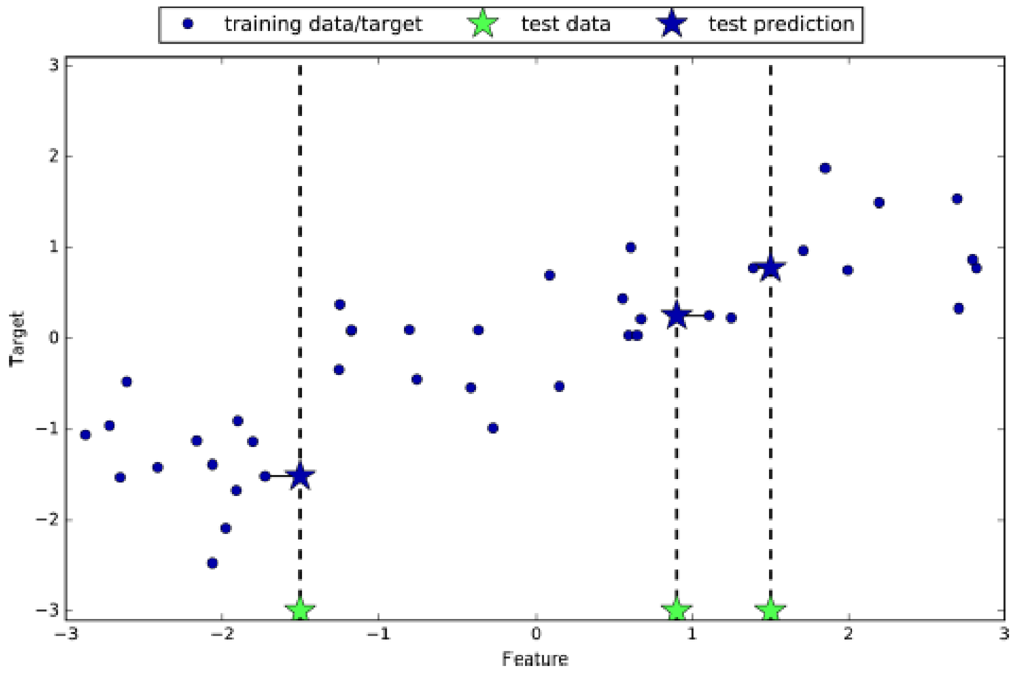


10-fold cross validation tells us that $K = 7$ results in the lowest validation error.

https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/

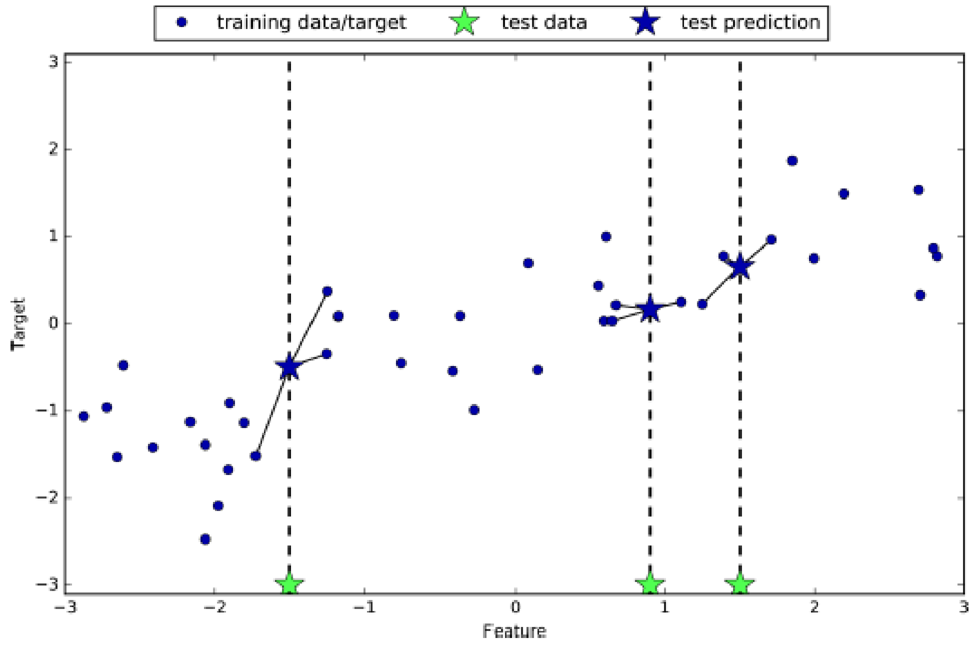
k-neighbors regression

- Let’s start by using the single nearest neighbor, this time using the wave dataset.
- We’ve added three test data points as green stars on the x-axis.
- The prediction using a single neighbor is just the target value of the nearest neighbor.
- These are shown as blue stars

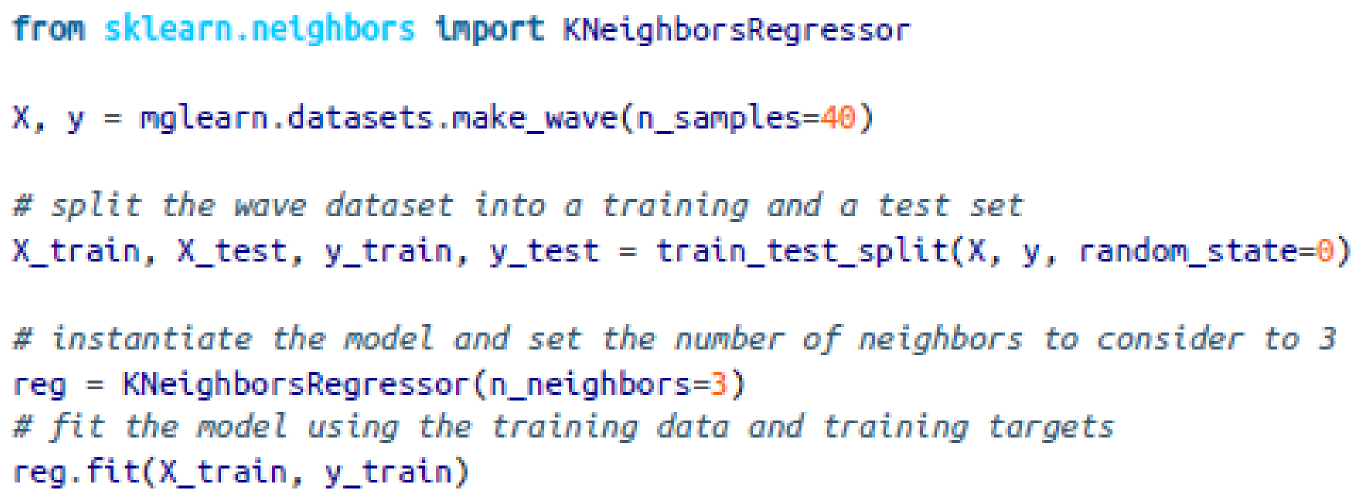


k-neighbors regression

When using multiple nearest neighbors, the prediction is the average, or mean, of the relevant neighbors



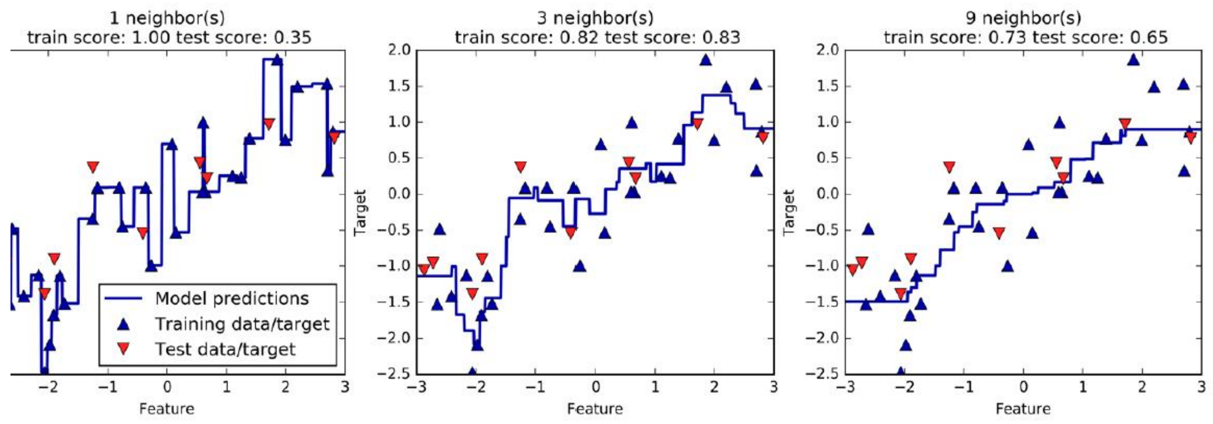
Scikit learn



```python
from sklearn.neighbors import KNeighborsRegressor

X, y = mglearn.datasets.make_wave(n_samples=40)

# split the wave dataset into a training and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# instantiate the model and set the number of neighbors to consider to 3
reg = KNeighborsRegressor(n_neighbors=3)
# fit the model using the training data and training targets
reg.fit(X_train, y_train)
```

Comparing predictions made by nearest neighbors regression for different values of n_neighbors

```python
fig, axes = plt.subplots(1, 3, figsize=(15, 4))
# create 1,000 data points, evenly spaced between -3 and 3
line = np.linspace(-3, 3, 1000).reshape(-1, 1)
for n_neighbors, ax in zip([1, 3, 9], axes):
    # make predictions using 1, 3, or 9 neighbors
    reg = KNeighborsRegressor(n_neighbors=n_neighbors)
    reg.fit(X_train, y_train)
    ax.plot(line, reg.predict(line))
    ax.plot(X_train, y_train, '^', c=mglearn.cm2(0), markersize=8)
    ax.plot(X_test, y_test, 'v', c=mglearn.cm2(1), markersize=8)

    ax.set_title(
        "{} neighbor(s)\n train score: {:.2f} test score: {:.2f}".format(
            n_neighbors, reg.score(X_train, y_train),
            reg.score(X_test, y_test)))
    ax.set_xlabel("Feature")
    ax.set_ylabel("Target")
axes[0].legend(["Model predictions", "Training data/target",
                "Test data/target"], loc="best")
```



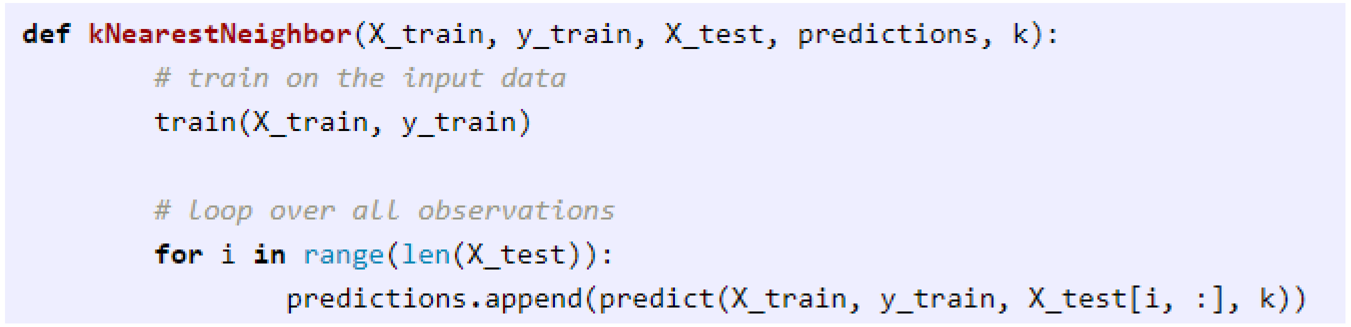
Writing our Own KNN from Scratch

```python
def train(X_train, y_train):
        # do nothing
        return
```

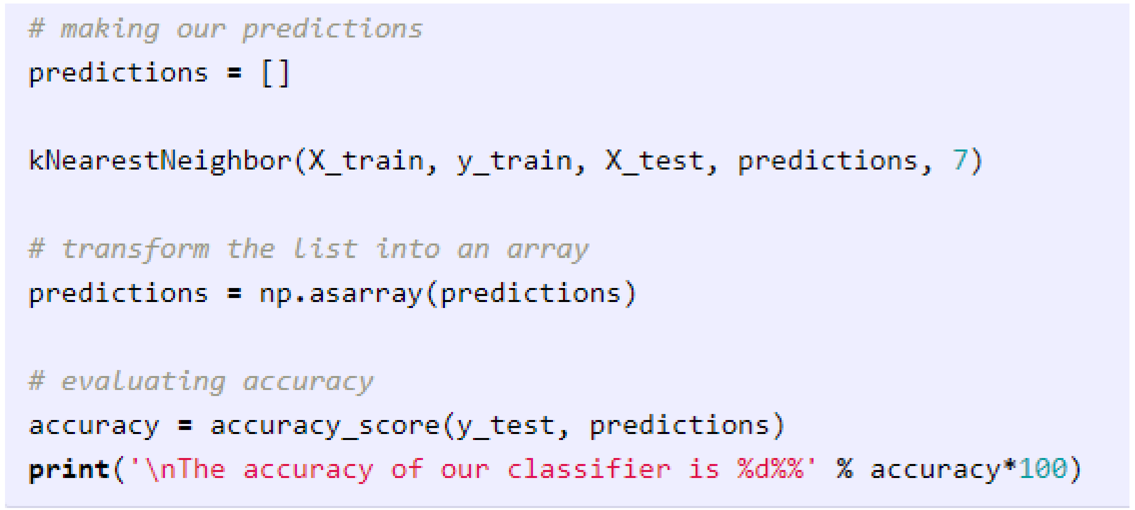
```python
def predict(X_train, y_train, x_test, k):
        # create list for distances and targets
        distances = []
        targets = []

        for i in range(len(X_train)):
                # first we compute the euclidean distance
                distance = np.sqrt(np.sum(np.square(x_test - X_train[i, :])))
                # add it to list of distances
                distances.append([distance, i])

        # sort the list
        distances = sorted(distances)

        # make a list of the k neighbors' targets
        for i in range(k):
                index = distances[i][1]
                targets.append(y_train[index])

        # return most common target
        return Counter(targets).most_common(1)[0][0]
```

https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/

Writing our Own KNN from Scratch



```python
def kNearestNeighbor(X_train, y_train, X_test, predictions, k):
        # train on the input data
        train(X_train, y_train)

        # loop over all observations
        for i in range(len(X_test)):
                predictions.append(predict(X_train, y_train, X_test[i, :], k))
```



```python
# making our predictions
predictions = []

kNearestNeighbor(X_train, y_train, X_test, predictions, 7)

# transform the list into an array
predictions = np.asarray(predictions)

# evaluating accuracy
accuracy = accuracy_score(y_test, predictions)
print('\nThe accuracy of our classifier is %d%%' % accuracy*100)
```

https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/