Семинар по метрическим методам

Теоретическая часть

Вспомнить из лекции:

- Как в методе k ближайших соседей выполняются предсказания в задаче классификаци и регрессии?
- Что такое гипотеза компактности?
- Какие функция расстояния можно использовать для вещественных признаков, категориальных признаков, строковых признаков, множественнозначных признаков?

Задача 1.

Предположим, мы решаем задачу классификации на три класса по двум признакам и используем метод k ближайших соседей с k=3 и манхэттанской метрикой. Мы имеем следующую обучающую выборку:

Признак 1	Признак 2	Класс
1	-1	1
2	2	1
3	2	2
1	0	3
2	-2	3

Каковы будут предсказания для объекта $x = (2, -1)$?

Решение.

Алгоритм предсказания kNN для задачи классификации:

1. Вычислить расстояние от каждого объекта обучающей выборки до тестового объекта.
2. Найти k объектов обучающей выборки (соседей) с наименьшим расстоянием до тестового объекта.
3. Вернуть наиболее встречающийся класс среди k соседей.

Вычислим расстояния. Расстояние от первого объекта в обучении до тестового объекта x (манхэттэнская метрика):

$$|1 - 2| + | - 1 - (-1)| = 1.$$

Аналогично для 2-5 объектов: получатся расстояния 3, 4, 2, 1.

Находим 3 ближайших объекта: это объекты с номерами 1, 4, 5 (расстояния 1, 2, 1 соответственно). Эти три объекта относятся к классам 1, 3, 3. Чаще всего встречается класс 3, поэтому предсказываем 3.

Задача 2.

Визуализируйте разделяющую поверхность между классами для следующей выборки:

Признак 1	Признак 2	Класс
2	2	1
3	2	1
2	0	2

Признак 1	Признак 2	Класс
1	-1	3
1	1	3

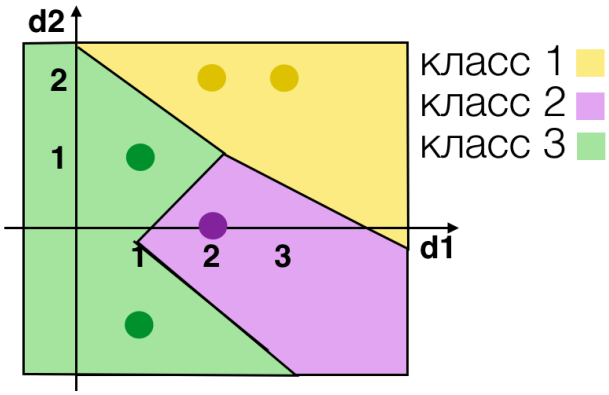
Используйте k=1 и евклидово расстояние.

Решение.

В задачах классификации с двумя признаками мы можем изобразить признаковое пространство на плоскости и раскрасить его в разные цвета в соответствии с классом каждой точки плоскости. В этом и состоит сейчас наша задача.

Для начала отобразим на плоскости обучающую выборку - пять точек - в соответствии с их координатами.

При $k = 1$ каждая точка плоскости будет относиться к тому же классу, что и ближайший к ней объект обучающей выборки. Если нам даны две точки разных классов, то чтобы провести между ними границу классов, нужно построить серединный перпендикудяр. Для случая с несколькими точками нужно построить несколько серединных перпендикуляров, найти их точки пересечения и определить, какие области к каким классам относятся. Более строго такая конструкция задается с помощью Диаграммы Вороного (https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_%D0 но мы не будем вдаваться в ее детали.



Задача 3.

Предположим, мы решаем задачу регрессии по двум признакам и используем метод k ближайших соседей с k=3 и манхэттанской метрикой. Мы имеем следующую обучающую выборку:

Признак 1	Признак 2	Ответ
1	-1	3.5
2	2	2.3
3	2	1.7
1	0	-0.4
2	-2	0.1

Каковы будут предсказания для объекта $x = (2, -1)$?

Решение. Предсказания kNN для регрессии отличаются от предсказаний для классификации только финальным шагом: вместо поиска наиболее часто встречающегося класса нужно усреднить ответы на соседях. Признаки в этой задаче те же, что в задаче 1, поэтому соседей мы уже знаем: это объекты с номерами 1, 4, 5. На них мы имеем ответы 3.5, -0.3, 0.1. Усредним их: (3.5-0.4+0.1)/3 = 1.1. Предсказываем 1.1.

Вопрос: каковы параметры и гиперпараметры метода kNN?

Ответ:

Параметры - это величины, которые мы настраиваем в процессе обучения по обучающей выборке. В методе kNN нет как такового обучения - это очень простой эвристический алгоритм. Под параметрами в kNN можно понимать обучающую выборку. В другой трактовке у метода нет параметров.

Гиперпараметры - это величины, которые мы должны установить до начала обучения модели. Гиперпараметры не настраиваются по обучающей выборке в процессе обучения модели. Два самых важных гиперпараметра метода kNN - это число соседей k и метрика. Используя разные комбинации этих гиперпараметров, можно получать совершенно разное качество работы алгоритма. Гиперпараметры обычно настраивают по валидационной выборке или используя кросс-валидацию.

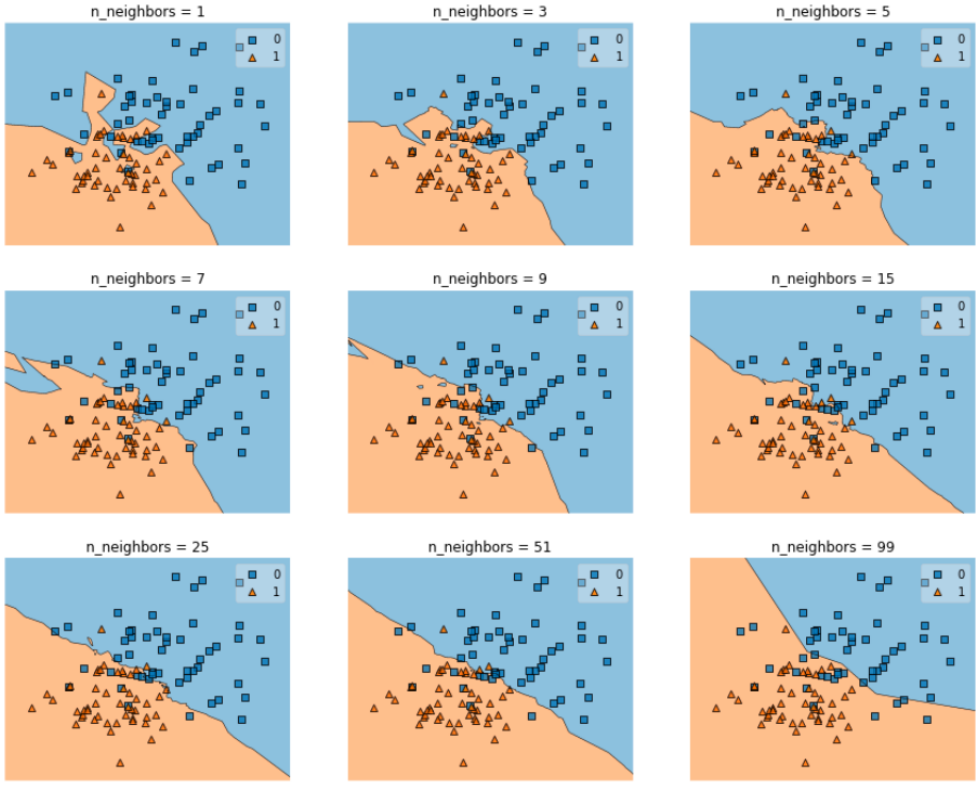
Какова динамика качества работы kNN при увеличении k?

Ответ:

При $k = 1$ вокруг каждого объекта обучающей выборки создается область его класса. Если, к примеру, в "большую" область одного класса случайно попал один шумовой объект другого класса, вокруг этого шумового объекта будет "остров" предсказания другого класса. Это нелогично и говорит о переобучении.

При k, равном числу объектов в выборке, для всех объектов будет предсказываться одно и то же, что вновь говорит о низком качестве работы классификатора. Получается, что качество kNN при увеличении k должно сначала расти, а потом падать, и оптимум будем где-то посередине.

Рассмотрим синтетический пример: на рисунке визуализирована обучающая выборка ("настоящая" разделяющая поверхность - прямая) и разделяющая поверхность kNN по аналогии с задачей 2, и на разных графиках используется разное число соседей k:



При использовании малых k разделяющая поверхность слишком сложная, на нее оказывают сильное воздействие шумовые объекты. Далее поверхность становится все ровнее и ровнее и при $k = 50$ выглядит наиболее разумно. При большем k разделяющая поверхность уходит от линейной, и оранжевый класс "захватывает" синий.

Почему при использовании kNN важно нормировать данные?

Ответ:

Рассмотрим для примера манхэттэнскую метрику. Если один признак будет иметь масштаб около 1000, а другой - около 1, то когда мы будем складывать модули разностей для этих двух признаков, второй признак практически не будет иметь влияния на ответ. Если же признаки отнормировать, но они все будут в одной шкале.

Интерфейс kNN описан в документации (http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html)

In [ ]:
```
import numpy as np
```

In [ ]:
```
import sklearn
```

In [38]:
```
# импортируем классификатор
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)
```

In [57]:
```
# загружаем данные --- изображения цифр
from sklearn.datasets import load_digits
data = load_digits()
```

In [58]:
```
X = data.images
y = data.target
```

In [59]:
```
X.shape # первое - число объектов, далее размер изображения 8 x 8
```

Out[59]:
```
(1797L, 8L, 8L)
```

In [60]:
```
X = X.reshape(X.shape[0], -1) # вытягиваем квадратное изображение в вектор, чтобы получить матрицу объекты-признаки
```

Чтобы оценить качество работы алгоритма на данных, которых не было в обучении, нужно разделить выборку на train и test. Помимо них, мы также выделим подвыборку val. На ней мы будем подбирать гиперпараметр k. Перемешаем объекты в случайном порядке и разделим выборку:

In [52]:
```
from sklearn.utils import shuffle
```

In [53]:
```
X, y = shuffle(X, y)
```

In [54]:
```
X.shape, y.shape # проверяем, что все хорошо перемешалось
```

Out[54]:
```
((1797L, 64L), (1797L,))
```

In [55]:
```
y[:10] # теперь в случайном порядке
```

Out[55]:
```
array([1, 4, 8, 6, 5, 2, 5, 6, 5, 6])
```

In [74]:
```
X_train, y_train = X[:700, :], y[:700]
X_val, y_val = X[700:1300, :], y[700:1300] #validation
X_test, y_test = X[1300:, :], y[1300:]
```

In [79]:
```
# Обучаем классификатор и делаем предсказания
clf.fit(X_train, y_train)
y_predicted = clf.predict(X_test)
```

In [80]:
```
# Вычисляем простейшую метрику качества алгоритма --- долю правильных ответов
print "Accuracy is", np.mean(y_test==y_predicted)
```

```
Accuracy is 0.955734406439
```

Учитывая, что у нас 10 классов, и вероятность случайно вытащить два одинаковых маленькая, точность 0.956 --- очень хороший результат!

Попробуем использовать разные значения гиперпараметра k. Сравнивать разные значения k по обучающей выборке бесполезно: каждый объект является ближайшим сам к себе6 и оптимальное k будет равно 1. Будем сравнивать разные k по качеству на валидационной выборке:

In [84]:
```
# Подбор k на валидационной выборке:
for k in range(1, 20):
    y_predicted = KNeighborsClassifier(n_neighbors=k).fit(X_train, y_train).predi
ct(X_val)
    print "k =", k, "; accuracy =", np.mean(y_predicted==y_val)
```

```
k = 1 ; accuracy = 0.958333333333
k = 2 ; accuracy = 0.956666666667
k = 3 ; accuracy = 0.956666666667
k = 4 ; accuracy = 0.956666666667
k = 5 ; accuracy = 0.958333333333
k = 6 ; accuracy = 0.958333333333
k = 7 ; accuracy = 0.963333333333
k = 8 ; accuracy = 0.965
k = 9 ; accuracy = 0.965
k = 10 ; accuracy = 0.966666666667
k = 11 ; accuracy = 0.958333333333
k = 12 ; accuracy = 0.961666666667
k = 13 ; accuracy = 0.96
k = 14 ; accuracy = 0.956666666667
k = 15 ; accuracy = 0.951666666667
k = 16 ; accuracy = 0.946666666667
k = 17 ; accuracy = 0.94
k = 18 ; accuracy = 0.94
k = 19 ; accuracy = 0.94
```

Наилучший результат при k=10

Сравним точность (accuracy) на обучении, валидации и тесте:

In [86]:
```
clf = KNeighborsClassifier(n_neighbors=10)
clf.fit(X_train, y_train)
for X_data, y_data in zip([X_train, X_val, X_test], [y_train, y_val, y_test]):
    y_predicted = clf.predict(X_data)
    print "Accuracy:", np.mean(y_predicted==y_data)
```

```
Accuracy: 0.987142857143
Accuracy: 0.966666666667
Accuracy: 0.941649899396
```

Качество на обучающей выборке самое лучшее, но оно обманчиво, ведь алгоритм уже знает эти объекты (переобучение). На валидационной выборке мы тоже использовали ответы --- уже для подбора гиперпараметра k, так что этот показатель тоже не совсем честный. И действительно, качество на тестовой выборке (ответы для которой мы нигде не подсматривали) оказалось хуже, чем на валидационной выборке.

Вывод: оценивать качество алгоритма нужно на отложенной выборке, которая не используется нигде в обучении и не используется в подборе гиперпараметров.