Check Questions (0.5 балла)

Вопрос 1: Объясните, чем отличается k-nearrest Neighbours от k-weighted nearest neighbours.

В k-nearrest Neighbours мы не учитываем расстояние от нашего объекта до его соседей, этот метод более подвержен неоднозначности классификации. Т.е пусть у нас k=2, первый сосед относится к классу 2 и находится ближе к объекту, который мы хотим классифицировать, а второй сосед относится к классу 1 и находится подальше. Тогда k-nearrest Neighbours выдаст коллизию, а k-weighted nearest neighbours покажет, что наш объект принадлежит второму классу, потому что первый сосед находится ближе.

Вопрос 2: Как изменяется абсолютное расстояние между объектами выборки при изменении метрики минковского с $p = 1$ ***до*** $p = \infty$***?***

$$\rho_1(x, y) = \sum_{i=1}^{d} |x_i - y_i|$$

$$\rho_\infty(x, y) = max_{i=1\ldots d}|x_i - y_i|$$

при $p = 1$ расстояние между объектами будет больше, чем при $p = \infty$, тк $max_{i=1\ldots d}|x_i - y_i| \leq \sum_{i=1}^{d} |x_i - y_i|$

Вопрос 3: Поясните, в чем суть проклятия размерности?

При росте размерности объекты становятся более удаленными друг от друга, т.е например чтобы с большой вероятностью найти несколько ближайших соседей точки $(0, 0, \ldots 0)$ среди равномерно распределенных точек в d-мерном кубе со стороной $[0; 1]$ нужно отступать на расстояния, быстро растущие с ростом d.

Вопрос 4: Что такое метрический отступ?

Пусть у нас есть объект $u \in X$. Введём понятие функции близости u к объектам класса $y \in Y$: $\Gamma_y(u)$ - просто складывает все объекты из класса y с их весами относительно u. А теперь на основе этой функции построим классификатор, который объекту сопоставляет класс с наибольший функцией близости:

Теперь оценим, насколько объект обучающей выборки "близок" своему классу. Для объекта x_i мы знаем, к какому классу он принадлежит - y_i, проверим, насколько хорошо наш объект подходит под свой класс:

$M(x_i) = \Gamma_{y_i}(x_i) - max_{y \in Y \setminus y_i}\Gamma_y(x_i)$ - мы взяли суммарный вес относительно x_i всех объетов из класса y_i и вычли максимальный вес объектов из другого класса, это называется метрический отступ. Чем больше эта величина, тем лучше объект подходит под свой класс

Вопрос 5: На какие типы можно разделить объекты обучающей выборки с точки значения значения метрического отступа? Какие объекты стоит исключить из выборки?

Для каждого объекта выборки оценим величину отступа, и отсортируем по этой величине. Получим пять условных классов объектов

Эталонные - объекты, которые хорошо подходят под свой класс, имеют большой положительный отступ. Можно брать за основу для классификации

Неинформативные - объекты, которые тоже имеют положительный отступ, но не добавляют никакой дополнительной информации к эталонным объектам. Наличие объектов такого класса характерно для избыточной выборки

Пограничные - с почти нулевым отступом, классификация таких объектов может измениться при изменении метрики или других каких-то параметров

Ошибочные - неверно классифицированные объекты. Ошибка может быть допущена из-за неудачного выбора модели

Шумовые объекты или выбросы - с большим отрицательным отступом. Т.е они окружены объектами другого класса. Такие объекты следует исключать из выборки.

Вопрос 6: Что такое функционал эмпирического риска? Приведите примеры.

Для того чтобы определить, насколько хорошо работает наш алгоритм, введём понятие $L(a, x)$ — функция потерь (насколько алгоритм a ошибается на $x \in X$).

Она может быть разной в зависимости от задачи:

$L(a, x) = I(x)$ - индикаторная функция, подходит для задач классификации.

$L(a, x) = |a(x) - y(x)|$ - показывает, на сколько ответ алгоритма отличается от настоящего. Для задачи регрессии.

А теперь усредним все "потери", получится эмпирическая функция риска:

$Q(a, X^l) = \frac{1}{l} \sum_{i=1}^{l} L(a, x_i)$ - таким образом, этот функционал показывает, насколько хорошо работает алгоритм на всей обучающей выборке. Нужно подобрать такой алгоритм a, чтобы значение функционала было как можно меньше

Вопрос 7: В чём суть явления переобучения?

Явление переобучения состоит в том, что наш алгоритм слишком подстроился под обучающую выборку, но не открыл общей закономерности. Или, что то же самое, эмпирическая функция мала на данных из обучающей выборки, но на контрольной выборке принимает большие значения.

Вопрос 8: Напишите формулу для complete cross-validation.

Суть метода complete cross validation состоит в том, чтобы разбить всю нашу выборку размера L на части размера l всеми способами, т.е $N = C_L^l, X = X_n^l \cup X_n^k, n \in \{1 \dots N\}, k = L - l$. Для каждого $n \in \{1 \dots N\}$ обучим алгоритм a на обучающей выборке X_n^l, проверим на контрольной X_n^k и усредним ошибку по всем элементам из контрольной выборки, а теперь усредним значения по всем разбиениям:

$CCV(\mu, X^L) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{k}\sum_{x_i \in X_n^k} I\{a(x_i, X_n^l) \neq y_i\}$ - это будет оценка качества нашего алгоритма

Задача 1 (0.5 балла)

При каких значения параметра k (при близких к единице или при сильно больших единицы) в алгоритме kNN можно наблюдать эффект переобучения? Поясните свой ответ, опираясь на границы классов из sklearn-knn-surfaces.ipynb.

При k близких к 1 наблюдается эффект переобучения, это видно, потому что синие точки при $k = 1, 2$ попадают в красную область, потому что поблизости находится одна красная точечка, а если посмотреть на большее количество соседей, то видно, что точки должны быть синими

Реализуйте kNN (2 балла)

In [88]:

```
from sklearn import datasets, metrics
from scipy.spatial.distance import cosine, euclidean, minkowski
import pandas as pd
from random import shuffle
import warnings
import numpy as np

warnings.simplefilter("ignore")
```

In [89]:

```
class kNNClassifier():
    def __init__(self, n_estimators, metric):
        """
        Parameters
        ----------
        n_neighbours: int
            Число соседей

        metric: *alias
            метрика измерения расстояний

          """
        self.n_neighbours = n_estimators
        self.metric = metric

    def fit(self, X, y):
        """

        Parameters
        ----------
        X: 2d np.array
        y: 1d np.array
        """

        # Тут храните описание объектов обучающей выборки
        self.X_learn = X

        # Тут храните здесь ответы по каждому объекту обучающей выборки
        self.y_learn = y

        # будем также хранить количество классов
        self.num_classes = len(set(y))

        return self

    def predict(self, X):
        """

        Parameters
        ----------
        X: 2d np.array матрица объекты признаки на которых нужно сказать ответ

        Returns
        -------
        y_pred: 1d np.array, Вектор классов для каждого объекта
        """

        dist = [] # Храните тут расстояния до каждого элемента обучающей выборки

        # количество ближайших соседей из каждого класса для каждого элемента X
        neigh = np.zeros((X.shape[0], self.num_classes))

        for j in range(X.shape[0]):
            dist.append([])
            for i in range(self.X_learn.shape[0]):
                # =====================================
                # рассчитайте расстояние до каждого объекта обучающей выборки
                # =====================================
```

```
                # добавляем пару - расстояние, класс

                dist[j].append((self.metric(self.X_learn[i], X[j]), self.y_learn[i]))

            # сортируем по расстоянию
            dist[j].sort()

            # выбираем первые n_neighbours соседей
            dist[j] = dist[j][:self.n_neighbours]

            # считаем, сколько ближайших соседей из каждого класса имеет объект (dist[obj][
            for i in range(self.n_neighbours):
                neigh[j][dist[j][i][1]] += 1


        # =======================================
        # предскажите класс каждого из объектов
        # =======================================
        y_pred = [ np.argmax(neigh[i]) for i in range(X.shape[0])]

        return y_pred
```

In [95]:

```
# Запустите ваш алгоритм на данных Digit_recognizer

train = pd.read_csv('data/digit_recognizer/train.csv')

indices = list(range(train.shape[0]))
shuffle(indices)

# уменьшила значения, потому что очень долго считает
subtrain, subtest = train.ix[indices[:500]], train.ix[indices[500:1000]]

X_train, X_test= np.asarray(subtrain[list(range(1, train.shape[1]))]), np.asarray(subtest[1
Y_train, Y_test = np.asarray(subtrain[[0]]).ravel(), np.asarray(subtest[[0]]).ravel()
```

In [91]:

```
from sklearn.metrics import accuracy_score
# =======================================
# Обучите классификатор при k=3, 5, и 10
# =======================================
```

In [97]:

```
clf = kNNClassifier(3, euclidean)
clf.fit(X_train, Y_train)
print("k = 3 " + "accuracy on train data ", accuracy_score(clf.predict(X_train), Y_train),
      accuracy_score(clf.predict(X_test), Y_test))
```

```
k = 3 accuracy on train data  0.924  accuracy on test data  0.822
```

In [100]:

```
clf = kNNClassifier(5, euclidean)
clf.fit(X_train, Y_train)
print("k = 5 " + "accuracy on train data ", accuracy_score(clf.predict(X_train), Y_train),
      accuracy_score(clf.predict(X_test), Y_test))
```

```
k = 5 accuracy on train data  0.908  accuracy on test data  0.812
```

In [101]:

```
clf = kNNClassifier(10, euclidean)
clf.fit(X_train, Y_train)
print("k = 10 " + "accuracy on train data ", accuracy_score(clf.predict(X_train), Y_train),
      accuracy_score(clf.predict(X_test), Y_test))
```

```
k = 10 accuracy on train data  0.878  accuracy on test data  0.772
```

Кросс-валидация (2 балла)

Зависимость значения CV от размера тестовой выборки.

Продемонстрируйте экспериментально, как меняется значения CV в зависимости от k - количество объектов в тестовой выборке, если объём всех данных меняется и равен $l + k$ (т.е. количество объектов в обучающей выборке НЕ меняется). Размер обучающей выборки l взять таким, чтобы можно было посчитать значения CV при $k = 10 \cdot l$. Демонстрацию провести на данных из соревнования Digit Recognizer на Kaggle. В качестве алгоритма классификации можно взять sklearn.neighbors.KNeighborsClassifier(algorithm='ball_tree').

In [78]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cross_validation import cross_val_score
```

In [83]:

```
l = 900
subtrain = train.ix[indices[:l]]

X_train = np.asarray(subtrain[list(range(1, train.shape[1]))])
Y_train = np.asarray(subtrain[[0]]).ravel()
```

In [108]:

```
K = [int(0.3*l), int(0.5*l), l, 3*l, 5*l, 7*l, 10*l]

estimator = KNeighborsClassifier(algorithm='ball_tree')
```

In [110]:

```
for k in K:

    # Реализуем кросс валидацию
    scoring = []
    for i in range(5):
        indices = np.arange(0, train.shape[0])
        shuffle(indices)

        subtrain, subtest = train.ix[indices[:l]], train.ix[indices[l:(l+k)]]
        X_train, X_test= np.asarray(subtrain[np.arange(1, train.shape[1])]), np.asarray(sub
        Y_train, Y_test = np.asarray(subtrain[[0]]).ravel(), np.asarray(subtest[[0]]).ravel

        estimator.fit(X_train, Y_train)

        score = estimator.score(X_test, Y_test)
        scoring.append(score)

    scoring = np.array(scoring)
    print("k=" + str(k) + " scoring.mean()", '%.3lf' % scoring.mean(), "scoring.std() "'%.4
```

```
k=270 scoring.mean() 0.874 scoring.std() 0.0057
k=450 scoring.mean() 0.871 scoring.std() 0.0121
k=900 scoring.mean() 0.865 scoring.std() 0.0079
k=2700 scoring.mean() 0.869 scoring.std() 0.0072
k=4500 scoring.mean() 0.874 scoring.std() 0.0072
k=6300 scoring.mean() 0.868 scoring.std() 0.0047
k=9000 scoring.mean() 0.867 scoring.std() 0.0061
```

Вывод: После проведения нескольких экспериметов выяснилось, что значения среднего примерно одинаковые и не зависят от размеров тестовой выборки, а дисперсия уменьшается и далее стабилизируется

In [ ]:

```

```