МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Высшая школа информационных технологий и информационных систем

**ОТЧЕТ**

**“Распознавание рукописных символов”**

**Выполнила:Шарипова Лия,**

**группа 11-401**

**Проверил: Новиков Петр Андреевич**

**2016**

Содержание

[**Описание набора данных**](#_tjc3xalwsyqe)[**2**](#_tjc3xalwsyqe)

[**Начальные условия**](#_t4k5fc82hacj)[**3**](#_t4k5fc82hacj)

[**Описание преобразований совершенных с данными**](#_ytro42mpdi2c)[**3**](#_ytro42mpdi2c)

[**Применяемые методы машинного обучения**](#_4uhycjqkf807)[**4**](#_4uhycjqkf807)

[**Оценка качества**](#_6jd4btkkrabp)[**6**](#_6jd4btkkrabp)

[**Заключение**](#_xr5f34y70q2t)[**7**](#_xr5f34y70q2t)

[**Приложение.**](#_ye1oh9i2yq94)[**8**](#_ye1oh9i2yq94)

# Описание набора данных

Данные представляют собой две csv-таблицы

1. train.csv
2. test.csv

Они содержат черно-белые изображения рукописных цифр от 0 до 9. Каждое изображение размера 28 пикселей в ширину и 28 пикселей в высоту. 784 пикселя в целом. Каждый пиксель изображения имеет сопоставленное ему целочисленное значение (pixel-value) насыщенности черного цвета в нем, которое измеряется от 0 до 255.

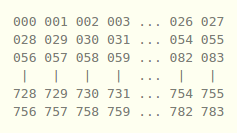
Данные для обучения представляют собой csv-таблицу с 785 колонками:

1. в первой колонке которой — численные значения написанных цифр (label)
2. в остальных — 784 значения насыщенности отдельно взятых пикселей (pixel-values)

Каждая колонка пикселей имеет название pixelx, где x = i \* 28 + j , i, j - (0, 27).

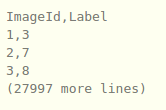
Пиксели расположены в матрице размера 28x28. Префикс pixel часто опускается.

Например, pixel31 расположен в 4м ряду и во 2й строке.



Данные для тестирования представляют собой ту же таблицу, что и данные для обучения, за исключением отсутствия колонки label, содержащей значение изображенной на картинке цифры.

На выходе мы должны получить файл, состоящий из 28000 строк. в котором для каждого изображения из test.csv указан id изображения и предполагаемое значение цифры.



# Начальные условия

Вычисления будут проводится на достаточно мощном компьютере с четырехъядерным процессором 3.2 Ггц, что обуславливает сравнительно быстрое время работы алгоритмов.

Для выполнения поставленной задачи будут применяться следующие методы:

1. Язык программирования Python
2. Библиотека машинного обучения Scikit-learn

Обоснование выбора используемых технологий:

Язык Python является одним из наиболее многофункциональных языков программирования на сегодняшний день. Наиболее распространенными инструментами Data Scientist'а на сегодняшний день являются R и Python.

В последнее время в машинном обучении Python набирает популярность благодаря хорошо задокументированной библиотеке scikit-learn.

# Описание преобразований совершенных с данными

Скачиваются наборы данных test.cvm и train.cvm с соревнования Digit Recognition.

Первым этапом необходимо преобразовать cvm таблицы в массивы данным, для этого используется метод  *genfromtxt()* библиотеки Numpy .

Далее данные подготавливаются к обучению, отделяются поля label (численное значение изображенного на картинке числа). На выходе два набора данных - train и target.

from numpy import genfromtxt *#для загрузки-сохранения csv-файлов*  
  
*#загружаем обучающие данные в многомерный массив*  
dataset = genfromtxt('train.csv', delimiter=',')

*#точно так же загружаем тестовые*  
test = genfromtxt('test.csv', delimiter=',')

target = [x[0] for x in dataset] *#метки*  
train = [x[1:] for x in dataset]

# Применяемые методы машинного обучения

Используемые алгоритмы

* Random Forest
* kNN-метод
* SVM с квадратичным ядром
* SVM с кубическим ядром

**Random Forest**

Random Forest использует ансамбль решающих деревьев.

Само по себе решающее дерево не обеспечивает достаточной точности для этой задачи, но отличается ***быстротой построения***.   
Алгоритм RF обучает k решающих деревьев на параметрах, случайно выбранных для каждого дерева (в нашем случае параметры — это яркости отдельных пикселей), после чего на каждом из тестов проводится голосование среди обученного ансамбля.

В основе построения этого алгоритма лежит идея о том, что если сагрегировать данные с большого количества различных слабых алгоритмов, сведя их в единый ответ, то результат, скорее всего, будет лучше, чем у одного мощного алгоритма.

***Выбранные параметры:***

Возникает проблема оптимального выбора числа деревьев: для данной задачи было сделано несколько прогонов с различными значениями n\_estimators, 1000 дала наилучший результат.

**kNN (k Nearest Neighbor )**

Один из наиболее быстрых алгоритмов классификации. В основе алгоритма: все сущности, используемые в обучении, помещаются в метрическое пространство размерности, равной количеству параметров. После чего при классификации отдельно взятого вектора рассматривается k векторов из обучающей выборки, наиболее близких к исследуемому.

Основной вопрос при использовании данного алгоритма — выбор числа k. При k = 1 алгоритм теряет устойчивость и начинается себя неадекватно вести при появлении шумов, при k, близком к числу векторов обучающей выборки, точность становится избыточной и алгоритм вырождается.

***Выбранные параметры:***

Алгоритм был запущен при разных k и в итоге результат был наилучшим при изначально рекомендованном k=10

**SVM**

SVM (Support Vector Machine) — один из наиболее универсальных методов классификации, отличающийся быстродействием и высокой надежностью.

SVM классифицирует векторы, расположенные в многомерном пространстве, схожем с тем, что используется в kNN, разделяя из гиперплоскостью, имеющую размерность n-1, где n — размерность исходного пространства.

***Выбранные параметры:***

Метод SVM имеет один из ключевых гиперпараметров, называемый ядром. В библиотеке scikit реализована поддержка всех основных используемых ядер: линейного, радиального и полиномиального.   
Статистику тестов на небольших выборках маркированных данных:

Точность с линейным ядром = 0.9131

Точность с радиальным ядром = 0.1265

Точность с квадратичным полиномом = 0.9635

Точность с кубическим полиномом = 0.9595

После этих тестов были варианты с линейным и радиальным ядрами.

Далее, на наибольшем возможном наборе данных сравнивались квадратичное и кубическое ядра. Квадратичное дало большую точность, оно и использовалось при оформлении финального решения.

# Оценка качества

Для оценки качества алгоритмов были рассмотрены следующие критерии:

* время
* точность

Для оценки точности был использован метод ***cross validation***

*#точность алгоритма*

scores = cross\_val\_score(Classificator, train, target, cv=5)

**print**("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() \* 2))

**Перекрёстная прове́рка** (англ. *cross-validation*) — метод оценки аналитической модели и её поведения на независимых данных. При оценке модели имеющиеся в наличии данные разбиваются на k частей. Затем на k−1 частях данных производится обучение модели, а оставшаяся часть данных используется для тестирования. Процедура повторяется k раз; в итоге каждая из k частей данных используется для тестирования. В результате получается оценка эффективности выбранной модели с наиболее равномерным использованием имеющихся данных.

Random Forest

это наиболее медленный алгоритм из рассмотренных.

time ~ 18 min ( 1098 sec)

accuracy = 0.91

KNN

kNN также показал точность выше 96% при меньшем времени работы.

time ~ 14 min ( 851 sec )

accuracy = 0.94

SVM

Работает достаточно быстро, с любым ядром обгоняет Random Forest,

точность наиболее высокая.

time ~ 17 min ( 1033 sec)

accuracy = 0.97

# Заключение

Для решения задачи Digit Recognition были рассмотрены алгоритмы классификации, такие как:

* Random Forest
* kNN-метод
* SVM с квадратичным ядром

В ходе исследования были получены следующие результаты:

* Random Forest показал среднюю точность (0.91) и среднее время выполнения (18 минут).
* kNN - самое быстрое время (14 минут), точность выше, чем у Random Forest (0.94).
* SVM - время примерно равное Random Forest (17 минут), точность наиболее высокая из вышеперечисленных (0.97).

Все алгоритмы показали относительно высокую точность. Если рассматривать приемлемость алгоритмов в определенных условиях:

* если *время не является решающим фактором*, для данной задачи наиболее приемлем алгоритм **SVM**.
* Если *важны и точность и время*, то в среднем наилучший результат показывает алгоритм **kNN**.

# Приложение.

Random Forest

from numpy import savetxt *#для сохранения csv-файлов*  
from sklearn.ensemble import RandomForestClassifier *# алгоритм классификации*  
from numpy import genfromtxt *#для загрузки csv-файлов*  
  
*#загружаем обучающие данные в многомерный массив*  
dataset = genfromtxt('train.csv', delimiter=',')

*#точно так же загружаем тестовые*  
test = genfromtxt('test.csv', delimiter=',')

*#отделяем метки от изображений*  
target = [x[0] for x in dataset] *#метки*  
train = [x[1:] for x in dataset]

*# n\_estimators - количество деревьев в ансамбле, n\_jobs - количество ядер, на которые распараллеливается алгоритм*  
forest = RandomForestClassifier(n\_estimators = 1000, n\_jobs = 4)   
  
*# замеряем время начала обучения и классификации*

start\_time = timeit.default\_timer()

forest.fit(train, target) *#обучаем деревья*

*#метод predict() возвращает многомерный массив (с которым работает numpy), в нашем случае - матрица 1x28000, которая записывается как столбец в csv-файл*  
pred = forest.predict(test)

*# время выполнения алгоритма*

elapsed = timeit.default\_timer() - start\_time

*#точность алгоритма*

scores = cross\_val\_score(forest, train, target, cv=5)

**print**("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() \* 2))

*#сохраняем полученные результаты в требуемом формате*  
savetxt('rf\_answer.csv', pred, delimiter=',', fmt='%d')

KNN

from sklearn import neighbors *# алгоритм классификации*  
from numpy import savetxt *#для загрузки-сохранения csv-файлов*  
from numpy import genfromtxt *#для загрузки csv-файлов*  
  
  
*#загружаем обучающие данные в многомерный массив*  
dataset = genfromtxt('train.csv', delimiter=',')

*#точно так же загружаем тестовые*  
test = genfromtxt('test.csv', delimiter=',')

*#инициализируем объект-классификатор*  
clf = neighbors.KNeighborsClassifier(n\_neighbors = 10, weights='uniform')

*# замеряем время начала обучения и классификации*

start\_time = timeit.default\_timer()  
clf.fit(train, target) *#обучаем*

pred = clf.predict(test)

*# время выполнения алгоритма*

elapsed = timeit.default\_timer() - start\_time  
*#точность алгоритма*

scores = cross\_val\_score(clf, train, target, cv=5)

**print**("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() \* 2))

*#сохраняем полученные результаты в требуемом формате*  
savetxt('knn\_answer.csv', pred, delimiter=',', fmt='%d')

SVM

from numpy import savetxt, loadtxt *#для загрузки-сохранения csv-файлов*  
from sklearn import svm *# алгоритм классификации*  
from numpy import genfromtxt *#для загрузки-сохранения csv-файлов*  
  
  
*#загружаем обучающие данные в многомерный массив*  
dataset = genfromtxt('train.csv', delimiter=',')

*#точно так же загружаем тестовые*  
test = genfromtxt('test.csv', delimiter=',')

*#инициализируем классификатор. Поле kernel указывает на ядро, degree - степень используемого полинома. По дефолту стоит 3.*  
clf\_poly2 = svm.SVC(kernel = "poly", degree = 2)

*# замеряем время начала обучения и классификации*

start\_time = timeit.default\_timer()  
clf\_poly2.fit(train, target) *#обучилем алгоритм*

pred = clf.predict(test)

*# время выполнения алгоритма*

elapsed = timeit.default\_timer() - start\_time  
*#точность алгоритма*

scores = cross\_val\_score(clf\_poly2, train, target, cv=5)

**print**("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() \* 2))

*#сохраняем полученные результаты в требуемом формате*  
savetxt('svm\_answer.csv', pred, delimiter=',', fmt='%d')