Федеральное государственное автономное образовательное учреждение

высшего образования

«Национальный исследовательский университет

«Высшая школа экономики»

Факультет компьютерных наук

ООП «Прикладная математика и информатика»

Отчёт о прохождении учебной практики

Студент: Харламов Алексей Владиславович

Группа: 153-1

Организация (место прохождения учебной практики): НИУ ВШЭ

Руководитель от организации

Старший преподаватель Факультета компьютерных наук / Кафедры технологий моделирования сложных систем, Янович Юрий Александрович

```
Москва, 2016
```

Задание

В работе предполагается применить стандартные алгоритмы классификации (методы ближайших соседей, байессовский классификатор, метод опорных векторов, решающие деревья и другие) для распознования рукописных цифр на примере популярной и общедоступной выборки данных MNIST (http://yann.lecun.com/exdb/mnist/). Также, применить методы нормализации данных (например, выравнивания угла изображения) и генерации признаков (например, метод главных компонент и графово-топологические) для улучшения качества классификации.

В ходе выполнения домашних работ предполагалось выполнить 4 задания:

Первое задание

1. Скачать выборку MNIST с сайта Kaggle или yann.lecun.com/exdb/mnist/.
2. Научиться загружать, записывать, отрисовывать, перевод из линейного вектора в двумерный для каждого предоставленного изображения.
3. Посчитать количество изображений во в выборке, число признаков, количество классов и объектов в каждом из них, значение признаков.
4. Средняя картинка (арифметическое по каждому пикселю) + средняя картинка по каждому классу.
5. Средне-квадратическая картинка.
6. Значимость пикселей. Есть ли абсолютно и почти белые и черные пиксели, придумать как отранжировать пиксели по значимости для классификации.

Второе задание

Во втором задании предлагалась восопльзоваться различными методами классификации, такими как:

1. KNN
2. SVM
3. Decision Tree
4. Neuron model

Нарисовать график зависимостей параметра для первых 2 методов.

Третье задание

1. Стандартизовать картинки (проверить центровку, избавиться от наклона (нарисовать картинки до выравнивания и после(определение угла и поворот))).
2. Применить 4 рассказанных метода (из предыдущего задания) для сравнения качества предсказательной модели.
3. Использовать замыкания и размыканная изображений и скелеты (используя алгоритм Розенфилда).
4. Сгенерить новые features (например запас связности, отношение длины к ширине) посмотреть насколько важны были эти признаки в классификаторе.

Четвертое задание

1. Нарисовать график зависимости доли первых n собственных чисел от всей суммы для PCA.
2. Выбрать картинки цифр, и нарисовать сжатие и растяжение, изучить как будет ухудшаться качество.
3. Нарисовать собственные векторы для трехмерной модели.
4. Применить алгоритмы классификации, посмотреть как изменится качество предсказательной модели с низкими размерностями.
5. Попробовать нелинейное снижение размерности, изучить получаемое качество с помощью классификаторов.
6. Отправить свой лучший результат на Kaggle.

Первое домашнее задание

In [3]:

```
import pandas
import matplotlib as plt
%matplotlib inline
import csv
import seaborn as sls
import numpy as np
from scipy.stats import kendalltau
from time import time
from sklearn.ensemble import ExtraTreesClassifier
from sklearn import cross_validation
from sklearn.metrics import accuracy_score

def log_progress(sequence, every=None, size=None):
```

```
    from ipywidgets import IntProgress, HTML, VBox
    from IPython.display import display

    is_iterator = False
    if size is None:
        try:
            size = len(sequence)
        except TypeError:
            is_iterator = True
    if size is not None:
        if every is None:
            if size <= 200:
                every = 1
            else:
                every = size / 200     # every 0.5%
    else:
        assert every is not None, 'sequence is iterator, set every'

    if is_iterator:
        progress = IntProgress(min=0, max=1, value=1)
        progress.bar_style = 'info'
    else:
        progress = IntProgress(min=0, max=size, value=0)
    label = HTML()
    box = VBox(children=[label, progress])
    display(box)

    index = 0
    try:
        for index, record in enumerate(sequence, 1):
            if index == 1 or index % every == 0:
                if is_iterator:
                    label.value = '{index} / ?'.format(index=index)
                else:
                    progress.value = index
                    label.value = u'{index} / {size}'.format(
                        index=index,
                        size=size
                    )
            yield record
    except:
        progress.bar_style = 'danger'
        raise
    else:
        progress.bar_style = 'success'
        progress.value = index
        label.value = str(index or '?')
```

In [4]:

```
train_set = pandas.read_csv("train.csv")
train_set
```

Out[4]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
6	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
7	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
8	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
10	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
11	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
12	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
13	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
14	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
15	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
16	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
18	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
19	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
20	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
21	6	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
22	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
24	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
25	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
26	6	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
27	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
28	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
29	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
41970	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41971	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41972	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41973	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41974	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41975	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41976	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41977	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41978	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41979	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41980	7	0	0	0	0	0	0	0	0	0	...	27	253	110	0	0	0	0	0	0	0
41981	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41982	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41983	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41984	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41985	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41986	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41987	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41988	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41989	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41990	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41991	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41992	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41993	6	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41994	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41996	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41997	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41998	6	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41999	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

42000 rows × 785 columns

In [4]:

```
test_set = train_set[train_set.pixel153 == 0.0]
test_set.to_csv('test_save.csv')
```

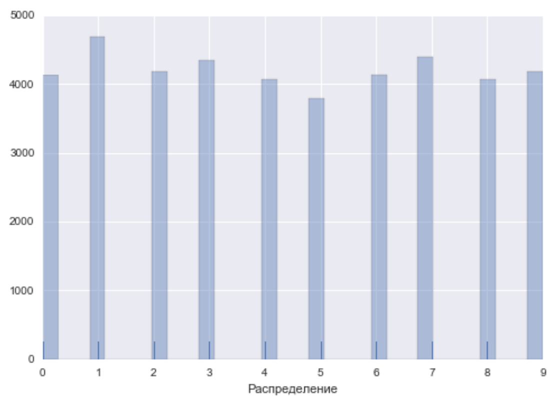
Имеем в тестовой выборке 41999 экземпляра для обучения, каждый из которых имеет по 785 параметров, где label - цифра, pixel0 - pixel783 значения соответсвующих пикселей в картинке.

In [5]:

```
graph = sls.distplot(train_set['label'],kde=False, rug=True)
graph.set(xlabel = 'Распределение')
graph
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x29d774a54e0>
```



In [6]:

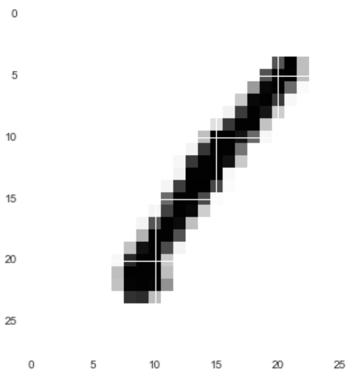
```
print(train_set['label'].describe())
print()
for i in range(10):
    print('we have', train_set[train_set.label == i].size // 785, 'of', i, 'images')
```

```
count    42000.000000
mean         4.456643
std          2.887730
min          0.000000
25%          2.000000
50%          4.000000
75%          7.000000
max          9.000000
Name: label, dtype: float64
```

```
we have 4132 of 0 images
we have 4684 of 1 images
we have 4177 of 2 images
we have 4351 of 3 images
we have 4072 of 4 images
we have 3795 of 5 images
we have 4137 of 6 images
we have 4401 of 7 images
we have 4063 of 8 images
we have 4188 of 9 images
```

In [5]:

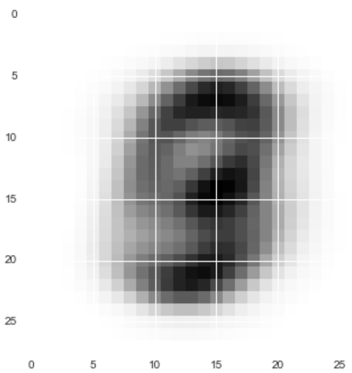
```
frame = train_set[:1]

def convert(frame):
    return frame.iloc[:,1:].values

def show_pic(img):
    img = img.reshape(28,28)
    plt.pyplot.imshow(img, interpolation='nearest')

show_pic(convert(frame))

def save_pic(img):
    plt.pyplot.savefig('img.jpg')

save_pic(convert(frame))
```



Средняя картинка попиксельно

In [8]:

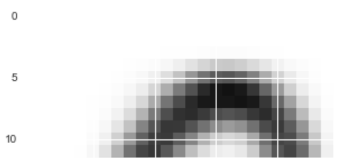
```
def deep_copy(self):
    return pandas.DataFrame(self.values.copy(), self.index.copy(), self.columns.copy())

mean_pic_pixel = deep_copy(train_set[:1])

for i in range(784):
    mean_pic_pixel['pixel' + str(i)] = int(train_set['pixel' + str(i)].mean())
show_pic(convert(mean_pic_pixel))
```

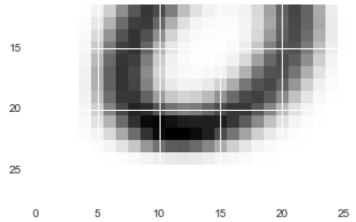


Средние картинки для каждого класса

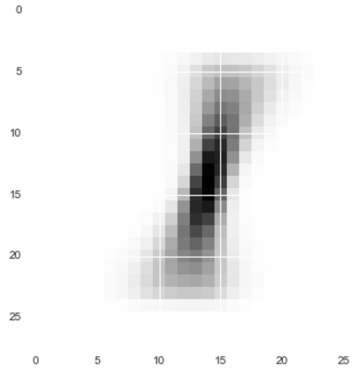
In [10]:

```
mean_pics = []
for i in range(10):
    mean_pics.append(deep_copy(train_set[:1]))

for j in range(10):
    tempset = train_set[train_set.label == j]
    for i in range(784):
        mean_pics[j]['pixel' + str(i)] = int(tempset['pixel' + str(i)].mean())
    fig = plt.pyplot.figure()
    show_pic(convert(mean_pics[j]))
    plt.pyplot.xlabel('Цифра ' + str(j))
```

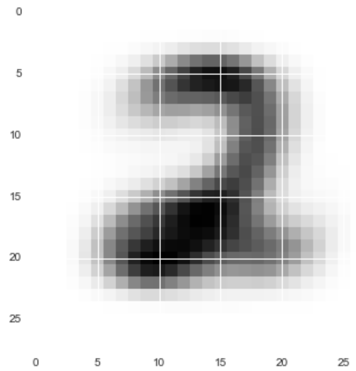




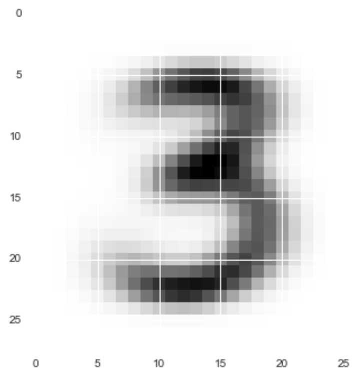
Цифра 0



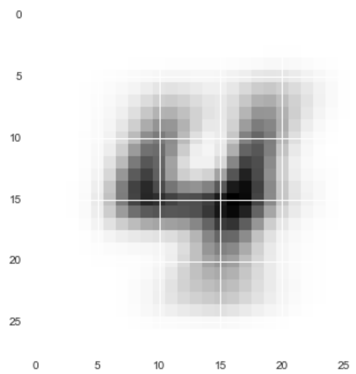
Цифра 1



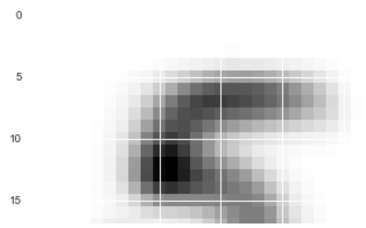
Цифра 2

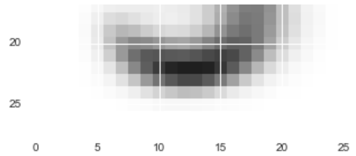


Цифра 3

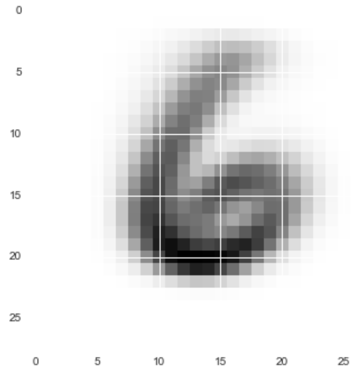


Цифра 4



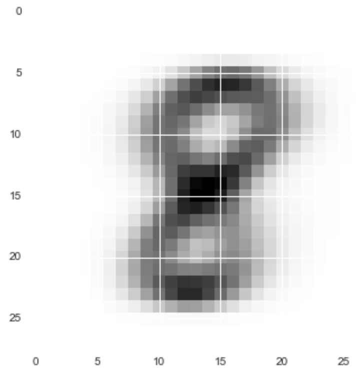


Цифра 5

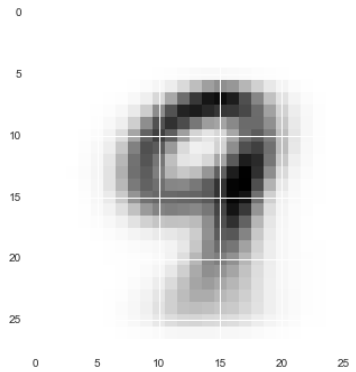


Цифра 6

Цифра 7



Цифра 8



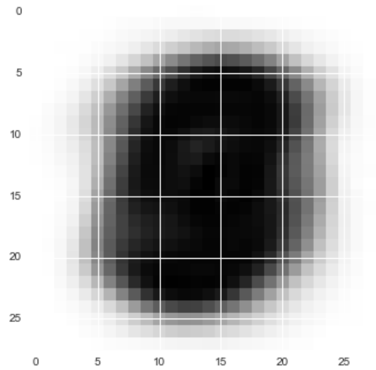
Цифра 9

Среднеквадратичная картинка по всей выборке

In [11]:

```
mean_sqr_pic_pixel = deep_copy(train_set[:1])

for i in range(784):
    mean = int(train_set['pixel' + str(i)].mean())
    ans = 0
    for elem in train_set['pixel' + str(i)]:
        ans += (mean - elem) ** 2
    ans /= len(train_set['pixel' + str(i)])
    ans = ans ** 0.5
    mean_sqr_pic_pixel['pixel' + str(i)] = ans
```

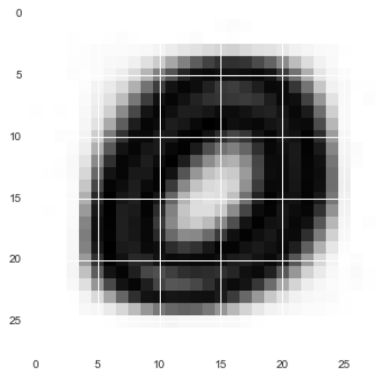
```
show_pic(convert(mean_sqr_pic_pixel))
```



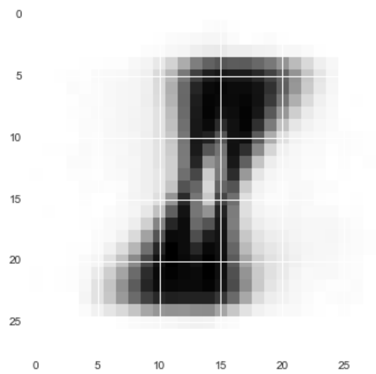
Среднеквадратичные картинки по каждому классу

In [12]:

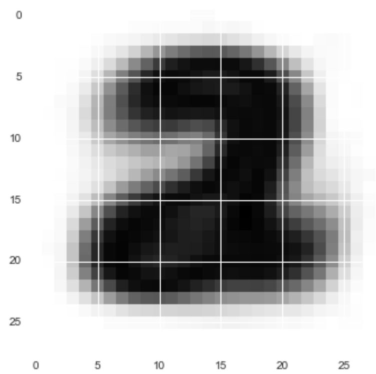
```
mean_sqr_pics = []
for i in range(10):
    mean_sqr_pics.append(deep_copy(train_set[:1]))

for j in range(10):
    tempset = train_set[train_set.label == j]
    for i in range(784):
        mean = int(tempset['pixel' + str(i)].mean())
        ans = 0
        for elem in tempset['pixel' + str(i)]:
            ans += (mean - elem) ** 2
        ans /= len(tempset['pixel' + str(i)])
        ans = ans ** 0.5
        mean_sqr_pics[j]['pixel' + str(i)] = ans
    fig = plt.pyplot.figure()
    show_pic(convert(mean_sqr_pics[j]))
    plt.pyplot.xlabel('Цифра ' + str(j))
```



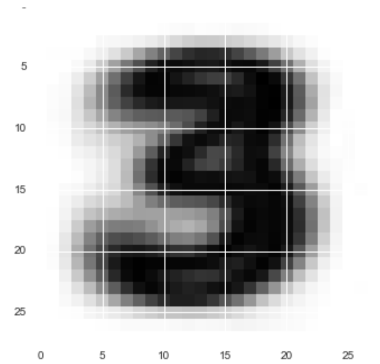
Цифра 0



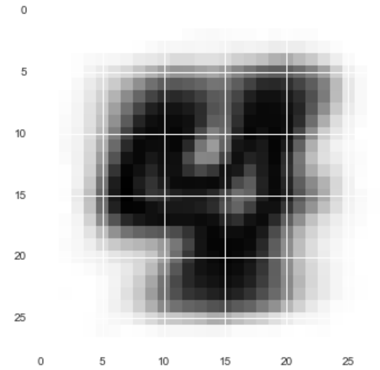
Цифра 1



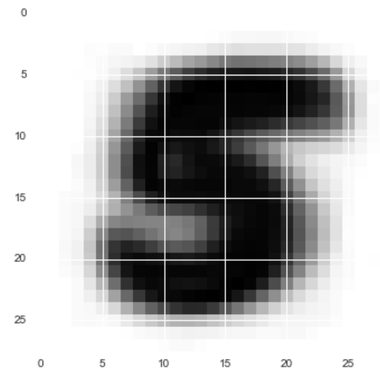
Цифра 2



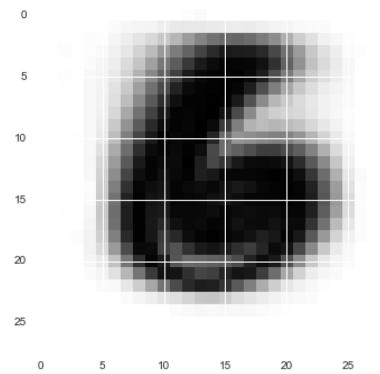
Цифра 3



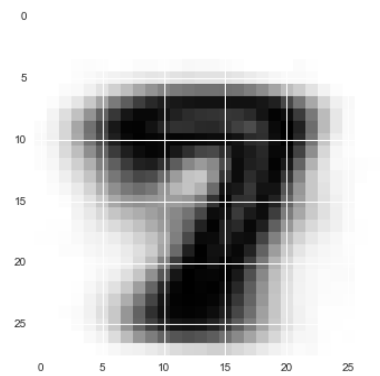
Цифра 4



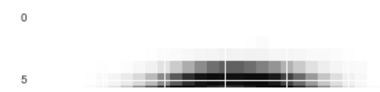
Цифра 5

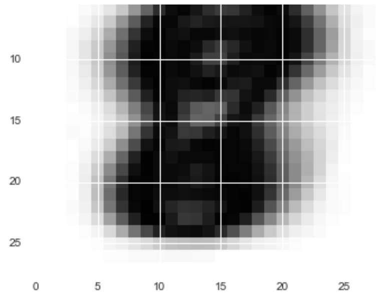


Цифра 6

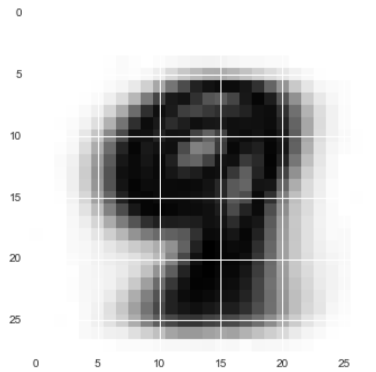


Цифра 7





Цифра 8



Цифра 9

Попробуем отранжировать пиксели по их значимости

In [33]:

```
white_pixel_stats = []
for row in train_set.columns[1:]:
    white_pixel_stats.append(len(train_set[train_set[row] == 0]))
black_pixel_stats = []
for row in train_set.columns[1:]:
    black_pixel_stats.append(len(train_set[train_set[row] == 255]))
```

In [34]:

```
pandas.DataFrame(white_pixel_stats).describe()
```

Out[34]:

	0
count	784.000000
mean	33955.755102
std	9781.264152
min	11549.000000
25%	25164.500000
50%	39745.000000
75%	41951.250000
max	42000.000000

In [35]:

```
pandas.DataFrame(black_pixel_stats).describe()
```

Out[35]:

	0
count	784.000000
mean	284.834184
std	378.719781
min	0.000000
25%	1.000000
50%	62.000000
75%	574.000000
max	1892.000000

Попробуем классификаторы для ранжирования пикселей

In [15]:

```
y = train_set.label
x = []
for i in range(42000):
    x.append(list(convert(train_set[i:i + 1])[0]))
```

In [62]:

```
n_jobs = -1

print("Started to fitting ExtraTreesClassifier on data with %d cores..." % 8 if n_jobs == -1 else n_jobs)
```

```
t0 = time()

forest = ExtraTreesClassifier(n_estimators=2000,
                              max_features=128,
                              n_jobs=n_jobs,
                              random_state=0)

X_train, X_test, y_train, y_test = cross_validation.train_test_split(x, y, test_size=0.2, random_state=0)

forest.fit(X_train, y_train)

y_predict = forest.predict(X_test)

print('we have ', accuracy_score(y_test, y_predict), 'of accuracy on ExtraTreeClassifier')

print("done in %0.3fs" % (time() - t0))

importances = forest.feature_importances_
```

```
Started to fitting ExtraTreesClassifier on data with 8 cores...
we have  0.971547619048 of accuracy on ExtraTreeClassifier
done in 497.239s
```

И так, имеем топ 15 в порядке убывания пикселей:

In [67]:

```
indices = np.argsort(importances)[::-1]

print("Feature ranking:")

for f in range(np.array(x).shape[1])[:15]:
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))
```

```
Feature ranking:
1. feature 378 (0.016453)
2. feature 350 (0.015496)
3. feature 461 (0.009529)
4. feature 211 (0.009148)
5. feature 489 (0.008974)
6. feature 409 (0.008961)
7. feature 542 (0.008364)
8. feature 406 (0.008251)
9. feature 462 (0.007785)
10. feature 514 (0.007740)
11. feature 433 (0.007735)
12. feature 155 (0.007684)
13. feature 210 (0.007522)
14. feature 375 (0.007499)
15. feature 347 (0.007446)
```

Второе домашнее задание

In [2]:

```
import pandas
import matplotlib as plt
%matplotlib inline
import csv
import seaborn as sls
from scipy.stats import kendalltau
from time import time
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn import cross_validation
from sklearn.metrics import accuracy_score
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
import os
import sys
import numpy
from six.moves import xrange
import tensorflow as tf
from sklearn import neighbors
from sklearn.svm import SVC
from tpot import TPOT
from sklearn.neural_network import MLPClassifier

def log_progress(sequence, every=None, size=None):
    from ipywidgets import IntProgress, HTML, VBox
    from IPython.display import display

    is_iterator = False
    if size is None:
        try:
            size = len(sequence)
        except TypeError:
            is_iterator = True
    if size is not None:
        if every is None:
            if size <= 200:
                every = 1
            else:
                every = size / 200     # every 0.5%
    else:
        assert every is not None, 'sequence is iterator, set every'

    if is_iterator:
        progress = IntProgress(min=0, max=1, value=1)
```

```
        progress.bar_style = 'info'
    else:
        progress = IntProgress(min=0, max=size, value=0)
    label = HTML()
    box = VBox(children=[label, progress])
    display(box)

    index = 0
    try:
        for index, record in enumerate(sequence, 1):
            if index == 1 or index % every == 0:
                if is_iterator:
                    label.value = '{index} / ?'.format(index=index)
                else:
                    progress.value = index
                    label.value = u'{index} / {size}'.format(
                        index=index,
                        size=size
                    )
            yield record
    except:
        progress.bar_style = 'danger'
        raise
    else:
        progress.bar_style = 'success'
        progress.value = index
        label.value = str(index or '?')
```

In [3]:

```
train_set = pandas.read_csv("train.csv")
```

In [4]:

```
def convert(frame):
    return frame.iloc[:,1:].values


def show_pic(img):
    img = img.reshape(28,28)
    plt.pyplot.imshow(img, interpolation='nearest')
```

In [5]:

```
y = train_set.label
x = []
for i in range(42000):
    x.append(list(convert(train_set[i:i + 1])[0]))

X_train, X_test, y_train, y_test = cross_validation.train_test_split(x, y, test_size=0.2, random_state=0)
```

In [ ]:

```
def kaggle_sub(sample, test_predict):
    for i in range(28000):
        sample.Label[i] = test_predict[i]
    sample.to_csv('submit.csv', columns=['ImageId', 'Label'], index=False)
kaggle_sub(sample,test_predict)
```

KNN с различными количествами соседей от 1 до 20 и одинаковой ценностью каждого соседа.

In [5]:

```
n_jobs = -1

print("Started to fitting KNN on data with %d cores..." % 8 if n_jobs == -1 else n_jobs)

accuracy_values = []
for number in range(1,20):
    t0 = time()

    clf = neighbors.KNeighborsClassifier(number, n_jobs=n_jobs, weights='uniform')

    clf.fit(X_train, y_train)

    y_predict = clf.predict(X_test)

    temp_acc = accuracy_score(y_test, y_predict)
    print('We have ', temp_acc, 'of accuracy on KNN with ', number, ' neighbor')
    accuracy_values.append(temp_acc)
    print("done in %0.3fs" % (time() - t0))
    print()
```

```
Started to fitting KNN on data with 8 cores...
We have  0.970952380952 of accuracy on KNN with  1  neighbor
done in 86.683s

We have  0.964404761905 of accuracy on KNN with  2  neighbor
done in 85.193s

We have  0.969761904762 of accuracy on KNN with  3  neighbor
done in 83.734s

We have  0.967619047619 of accuracy on KNN with  4  neighbor
done in 84.356s

We have  0.968095238095 of accuracy on KNN with  5  neighbor
done in 86.186s

We have  0.967380952381 of accuracy on KNN with  6  neighbor
done in 85.625s

We have  0.967738095238 of accuracy on KNN with  7  neighbor
done in 101.367s

We have  0.967023809524 of accuracy on KNN with  8  neighbor
done in 99.815s
```

```
We have  0.966785714286 of accuracy on KNN with  9  neighbor
done in 106.013s

We have  0.965119047619 of accuracy on KNN with  10  neighbor
done in 106.116s

We have  0.963928571429 of accuracy on KNN with  11  neighbor
done in 112.595s

We have  0.962261904762 of accuracy on KNN with  12  neighbor
done in 112.811s

We have  0.960119047619 of accuracy on KNN with  13  neighbor
done in 111.461s

We have  0.959523809524 of accuracy on KNN with  14  neighbor
done in 109.430s

We have  0.959285714286 of accuracy on KNN with  15  neighbor
done in 107.924s

We have  0.959404761905 of accuracy on KNN with  16  neighbor
done in 106.823s

We have  0.958214285714 of accuracy on KNN with  17  neighbor
done in 91.176s

We have  0.957738095238 of accuracy on KNN with  18  neighbor
done in 91.910s

We have  0.95619047619 of accuracy on KNN with  19  neighbor
done in 94.237s
```

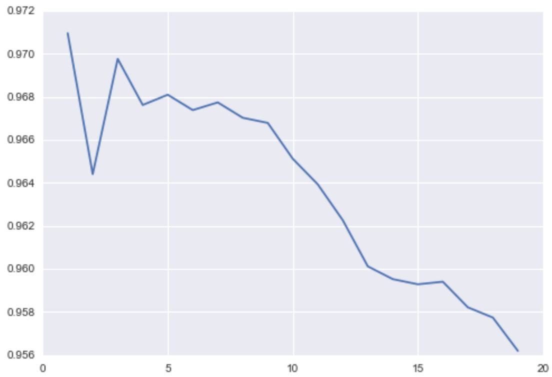
Распределение ошибки в зависимости от количества соседей

In [6]:

```python
x_neib = numpy.array([i for i in range(1,20)])
plt.pyplot.plot(x_neib,accuracy_values)
```

Out[6]:

```
[<matplotlib.lines.Line2D at 0x27d9ae357f0>]
```



KNN для различного количества соседей с изменением веса в зависимости от удаленности соседа

In [7]:

```python
n_jobs = -1

print("Started to fitting KNN on data with %d cores..." % 8 if n_jobs == -1 else n_jobs)

accuracy_values = []
for number in range(1,20):
    t0 = time()

    clf = neighbors.KNeighborsClassifier(number, n_jobs=n_jobs, weights='distance')

    clf.fit(X_train, y_train)

    y_predict = clf.predict(X_test)

    temp_acc = accuracy_score(y_test, y_predict)
    print('We have ', temp_acc, 'of accuracy on KNN with ', number, ' neighbor')
    accuracy_values.append(temp_acc)
    print("done in %0.3fs" % (time() - t0))
    print()
```

```
Started to fitting KNN on data with 8 cores...
We have  0.970952380952 of accuracy on KNN with  1  neighbor
done in 88.269s

We have  0.970952380952 of accuracy on KNN with  2  neighbor
done in 92.713s

We have  0.970714285714 of accuracy on KNN with  3  neighbor
done in 93.633s

We have  0.9725 of accuracy on KNN with  4  neighbor
done in 97.906s

We have  0.970357142857 of accuracy on KNN with  5  neighbor
done in 99.610s

We have  0.97119047619 of accuracy on KNN with  6  neighbor
done in 100.097s

We have  0.968571428571 of accuracy on KNN with  7  neighbor
```

```
We have  0.968571428571 of accuracy on KNN with  7  neighbor
done in 98.970s

We have  0.969523809524 of accuracy on KNN with  8  neighbor
done in 99.758s

We have  0.96869047619 of accuracy on KNN with  9  neighbor
done in 99.456s

We have  0.967261904762 of accuracy on KNN with  10  neighbor
done in 98.244s

We have  0.965714285714 of accuracy on KNN with  11  neighbor
done in 96.456s

We have  0.964880952381 of accuracy on KNN with  12  neighbor
done in 96.304s

We have  0.962619047619 of accuracy on KNN with  13  neighbor
done in 96.844s

We have  0.962261904762 of accuracy on KNN with  14  neighbor
done in 94.389s

We have  0.961071428571 of accuracy on KNN with  15  neighbor
done in 94.447s

We have  0.961428571429 of accuracy on KNN with  16  neighbor
done in 105.964s

We have  0.959404761905 of accuracy on KNN with  17  neighbor
done in 105.069s

We have  0.959880952381 of accuracy on KNN with  18  neighbor
done in 107.483s

We have  0.958333333333 of accuracy on KNN with  19  neighbor
done in 108.562s
```

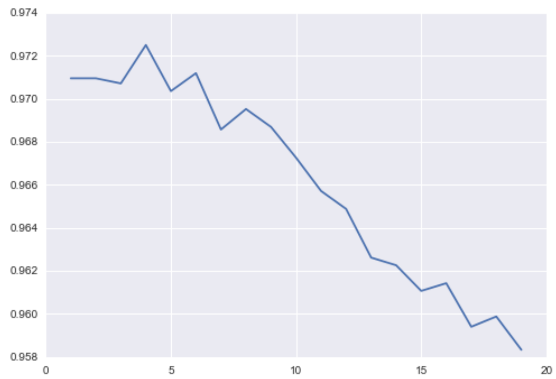
Распределение ошибки для зависимой от расстояния ценности соседа

In [8]:

```
x_neib = numpy.array([i for i in range(1,20)])
plt.pyplot.plot(x_neib,accuracy_values)
```

Out[8]:

```
[<matplotlib.lines.Line2D at 0x27df79fbeb8>]
```



SVM с различными коэффициентами

In [9]:

```
print("Started to fitting SVM on data")

accuracy_values_SVM = []
for number in [0.5, 0.7, 1]:
    t0 = time()

    clf = SVC(C=number)

    clf.fit(X_train, y_train)

    y_predict = clf.predict(X_test)

    temp_acc = accuracy_score(y_test, y_predict)
    print('We have ', temp_acc, 'of accuracy on SVC with C=', number)
    accuracy_values.append(temp_acc)
    print("done in %0.3fs" % (time() - t0))
    print()
```

```
Started to fitting SVM on data
We have  0.114404761905 of accuracy on SVC with C= 0.5
done in 3119.694s

We have  0.114404761905 of accuracy on SVC with C= 0.7
done in 3078.368s

We have  0.114404761905 of accuracy on SVC with C= 1
done in 2953.908s
```

Не имеем никакой разницы в зависимости от параметра =(

SVM с измененным ядром для полиномов от 1 до 10

In [10]:

```
print("Started to fitting poly SVM on data")

accuracy_values_poly_SVM = []
for number in range(1,10):
    t0 = time()

    clf = SVC(kernel='poly', degree=number)

    clf.fit(X_train, y_train)

    y_predict = clf.predict(X_test)

    temp_acc = accuracy_score(y_test, y_predict)
    print('We have ', temp_acc, 'of accuracy on poly SVC with degree =', number)
    accuracy_values_poly_SVM.append(temp_acc)
    print("done in %0.3fs" % (time() - t0))
    print()
```

```
Started to fitting poly SVM on data
We have  0.919047619048 of accuracy on poly SVC with degree = 1
done in 412.535s

We have  0.977738095238 of accuracy on poly SVC with degree = 2
done in 167.451s

We have  0.974047619048 of accuracy on poly SVC with degree = 3
done in 173.696s

We have  0.969166666667 of accuracy on poly SVC with degree = 4
done in 185.534s

We have  0.959761904762 of accuracy on poly SVC with degree = 5
done in 206.720s

We have  0.947619047619 of accuracy on poly SVC with degree = 6
done in 234.237s

We have  0.932380952381 of accuracy on poly SVC with degree = 7
done in 268.058s

We have  0.919642857143 of accuracy on poly SVC with degree = 8
done in 306.511s

We have  0.903214285714 of accuracy on poly SVC with degree = 9
done in 347.078s
```

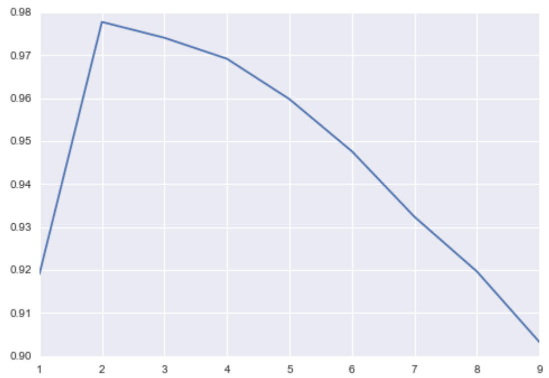
Точность в зависимости от различных ядер

In [11]:

```
x_neib = numpy.array([i for i in range(1,10)])
plt.pyplot.plot(x_neib,accuracy_values_poly_SVM)
```

Out[11]:

```
[<matplotlib.lines.Line2D at 0x27d8001ccf8>]
```



Обычное дерево решений

In [16]:

```
print("Started to fitting DecisionTreeClassifier on data")

t0 = time()

forest = DecisionTreeClassifier()

forest.fit(X_train, y_train)

y_predict = forest.predict(X_test)

print('we have ', accuracy_score(y_test, y_predict), 'of accuracy on DecisionTreeClassifier')

print("done in %0.3fs" % (time() - t0))
```

```
Started to fitting DecisionTreeClassifier on data
we have  0.8525 of accuracy on DecisionTreeClassifier
done in 12.799s
```

RandomForest с 2000 тысячами деревьев и голосованием для выбора предсказания.

In [8]:

```
n_jobs = -1
```

```python
print("Started to fitting ExtraTreesClassifier on data with %d cores..." % 8 if n_jobs == -1 else n_jobs)

t0 = time()

forest = ExtraTreesClassifier(n_estimators=2000,
                              max_features=128,
                              n_jobs=n_jobs,
                              random_state=0)

forest.fit(X_train, y_train)

y_predict = forest.predict(X_test)

print('we have ', accuracy_score(y_test, y_predict), 'of accuracy on ExtraTreeClassifier')

print("done in %0.3fs" % (time() - t0))
```

```
Started to fitting ExtraTreesClassifier on data with 8 cores...
we have  0.971547619048 of accuracy on ExtraTreeClassifier
done in 491.892s
```

Градиентный бустинг решающих деревьев

In [18]:

```python
print("Started to fitting GradientBoostingClassifier on data")

t0 = time()

forest = GradientBoostingClassifier()

forest.fit(X_train, y_train)

y_predict = forest.predict(X_test)

print('we have ', accuracy_score(y_test, y_predict), 'of accuracy on GradientBoostingClassifier')

print("done in %0.3fs" % (time() - t0))
```

```
Started to fitting GradientBoostingClassifier on data
we have  0.943928571429 of accuracy on GradientBoostingClassifier
done in 1382.034s
```

RandomForest с различным количеством деревьев

In [19]:

```python
print("Started to fitting RandomForest on data")

accuracy_values_random_forest = []
for number in [2, 10, 100, 300, 700, 1000, 2000]:
    t0 = time()

    clf = RandomForestClassifier(n_estimators=number, n_jobs=-1)

    clf.fit(X_train, y_train)

    y_predict = clf.predict(X_test)

    temp_acc = accuracy_score(y_test, y_predict)
    print('We have ', temp_acc, 'of accuracy on RandomForest with trees =', number)
    accuracy_values_random_forest.append(temp_acc)
    print("done in %0.3fs" % (time() - t0))
    print()
```

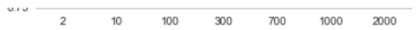
```
Started to fitting RandomForest on data
We have  0.784404761905 of accuracy on RandomForest with trees = 2
done in 2.902s

We have  0.934166666667 of accuracy on RandomForest with trees = 10
done in 3.185s

We have  0.964523809524 of accuracy on RandomForest with trees = 100
done in 7.170s

We have  0.966547619048 of accuracy on RandomForest with trees = 300
done in 17.586s

We have  0.966428571429 of accuracy on RandomForest with trees = 700
done in 36.240s

We have  0.965952380952 of accuracy on RandomForest with trees = 1000
done in 51.613s

We have  0.96619047619 of accuracy on RandomForest with trees = 2000
done in 108.641s
```

In [30]:

```python
x_neib = numpy.array([2, 10, 100, 300, 700, 1000, 2000])
sls.boxplot(x_neib,accuracy_values_random_forest)
```

Out[30]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x27d9a7d3f28>
```



AdaBoost с различных количеством решающих деревьев

In [31]:

```
print("Started to fitting AdaBoost on data")

accuracy_values_Ada = []
for number in [2, 10, 100, 300, 700, 1000, 2000]:
    t0 = time()

    clf = AdaBoostClassifier(n_estimators=number)

    clf.fit(X_train, y_train)

    y_predict = clf.predict(X_test)

    temp_acc = accuracy_score(y_test, y_predict)
    print('We have ', temp_acc, 'of accuracy on AdaBoost with trees =', number)
    accuracy_values_Ada.append(temp_acc)
    print("done in %0.3fs" % (time() - t0))
    print()
```

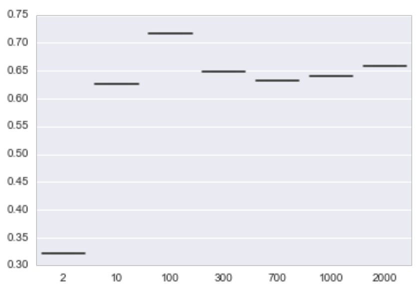
```
Started to fitting AdaBoost on data
We have  0.321547619048 of accuracy on AdaBoost with trees = 2
done in 4.522s

We have  0.62619047619 of accuracy on AdaBoost with trees = 10
done in 9.582s

We have  0.717857142857 of accuracy on AdaBoost with trees = 100
done in 77.965s

We have  0.648928571429 of accuracy on AdaBoost with trees = 300
done in 226.388s

We have  0.631666666667 of accuracy on AdaBoost with trees = 700
done in 501.802s

We have  0.640833333333 of accuracy on AdaBoost with trees = 1000
done in 685.638s

We have  0.658452380952 of accuracy on AdaBoost with trees = 2000
done in 1333.760s
```

In [32]:

```
x_neib = numpy.array([2, 10, 100, 300, 700, 1000, 2000])
sls.boxplot(x_neib,accuracy_values_Ada)
```

Out[32]:

<matplotlib.axes._subplots.AxesSubplot at 0x27d9a758668>



Генетический алгоритм подбора наилучших параметров

In [9]:

```
print('Started to fitting')
t0 = time()
tpot = TPOT(generations=10, verbosity=2)
tpot.fit(X_train, numpy.array(y_train))
print(tpot.score(X_test, y_test))
print("done in %0.3fs" % (time() - t0))

result1 = tpot_data.copy()

bnb1 = BernoulliNB(alpha=1e-05, binarize=0.03)
bnb1.fit(result1.loc[training_indices].drop('class', axis=1).values, result1.loc[training_indices, 'class'].values)

result1['bnb1-classification'] = bnb1.predict(result1.drop('class', axis=1).values)


training_features = result1.loc[training_indices].drop('class', axis=1)
training_class_vals = result1.loc[training_indices, 'class'].values

if len(training_features.columns.values) == 0:
    result2 = result1.copy()
else:
    selector = SelectKBest(f_classif, k=min(38, len(training_features.columns)))
    selector.fit(training_features.values, training_class_vals)
    mask = selector.get_support(True)
    mask_cols = list(training_features.iloc[:, mask].columns) + ['class']
    result2 = result1[mask_cols]
```

```
Started to fitting
```

```
Generation 1 - Current best internal CV score: 0.88261
```

```
Generation 2 - Current best internal CV score: 0.88261
```

```
Generation 3 - Current best internal CV score: 0.88261

Generation 4 - Current best internal CV score: 0.88261

Generation 5 - Current best internal CV score: 0.88261

Generation 6 - Current best internal CV score: 0.88261

Generation 7 - Current best internal CV score: 0.88261

Generation 8 - Current best internal CV score: 0.88261

Generation 9 - Current best internal CV score: 0.88261

Generation 10 - Current best internal CV score: 0.88261

Best pipeline: _select_kbest(_bernoulli_nb(input_df, 1.0000000000000001e-05, 0.029999999999999999), 38)
0.0
done in 555.827s
```

6 слойный перцептрон с параметрами 784-2500-2000-1500-1000-500-10 со стохастическим градиентным спуском и различными функциями активации и алгоритмами

In [12]:

```
print("Started to fitting MLPClassifier on data")

accuracy_values_MLPClassifier = []
for activate_type in ['logistic', 'tanh', 'relu']:
    for algo in ['sgd', 'adam', 'l-bfgs']:
        t0 = time()

        clf = MLPClassifier(verbose=True, activation=activate_type, learning_rate='adaptive',
                            hidden_layer_sizes=(2500, 2000, 1500, 1000, 500))

        clf.fit(X_train, y_train)

        y_predict = clf.predict(X_test)

        temp_acc = accuracy_score(y_test, y_predict)
        print('We have ', temp_acc,
              'of accuracy on MLPclassifier with activation function =',
              activate_type, 'with ', algo, 'algorythm')
        accuracy_values_MLPClassifier.append(temp_acc)
        print("done in %0.3fs" % (time() - t0))
        print()
```

```
Started to fitting MLPClassifier on data
Iteration 1, loss = 0.87510415
Iteration 2, loss = 0.31951016
Iteration 3, loss = 0.27244733
Iteration 4, loss = 0.23901066
Iteration 5, loss = 0.23766333
Iteration 6, loss = 0.21435707
Iteration 7, loss = 0.21254382
Iteration 8, loss = 0.19835075
Iteration 9, loss = 0.19636110
Iteration 10, loss = 0.17523291
Iteration 11, loss = 0.18589492
Iteration 12, loss = 0.17654913
Iteration 13, loss = 0.17054411
Iteration 14, loss = 0.16244333
Iteration 15, loss = 0.15656101
Iteration 16, loss = 0.16103789
Iteration 17, loss = 0.15590492
Iteration 18, loss = 0.15037203
Iteration 19, loss = 0.14901456
Iteration 20, loss = 0.14516383
Iteration 21, loss = 0.13650559
Iteration 22, loss = 0.13388478
Iteration 23, loss = 0.14489743
Iteration 24, loss = 0.13223027
Iteration 25, loss = 0.12863507
Iteration 26, loss = 0.14548051
Iteration 27, loss = 0.12500395
Iteration 28, loss = 0.12471742
Iteration 29, loss = 0.12805905
Iteration 30, loss = 0.12056053
Iteration 31, loss = 0.11958055
Iteration 32, loss = 0.11432730
Iteration 33, loss = 0.10925375
Iteration 34, loss = 0.11498923
Iteration 35, loss = 0.11486075
Iteration 36, loss = 0.10981611
Training loss did not improve more than tol=0.000100 for two consecutive epochs. Stopping.
We have  0.950476190476 of accuracy on MLPclassifier with activation function = logistic with  sgd algorythm
done in 4150.909s

Iteration 1, loss = 0.93831186
Iteration 2, loss = 0.33989846
Iteration 3, loss = 0.28095171
Iteration 4, loss = 0.25125090
Iteration 5, loss = 0.23357120
Iteration 6, loss = 0.21503037
Iteration 7, loss = 0.21098317
Iteration 8, loss = 0.19875734
Iteration 9, loss = 0.17656493
Iteration 10, loss = 0.17417047
Iteration 11, loss = 0.18163265
Iteration 12, loss = 0.16748495
Iteration 13, loss = 0.16500104
Iteration 14, loss = 0.17438896
Iteration 15, loss = 0.16209609
Iteration 16, loss = 0.15526526
Iteration 17, loss = 0.14461733
Iteration 18, loss = 0.15112696
Iteration 19, loss = 0.14075751
Iteration 20, loss = 0.14859385
Iteration 21, loss = 0.14786730
```

```
Iteration 22, loss = 0.13804434
Iteration 23, loss = 0.14639926
Iteration 24, loss = 0.13127357
Iteration 25, loss = 0.13043497
Iteration 26, loss = 0.12039773
Iteration 27, loss = 0.12296756
Iteration 28, loss = 0.12387849
Iteration 29, loss = 0.12365246
Training loss did not improve more than tol=0.000100 for two consecutive epochs. Stopping.
We have  0.940238095238 of accuracy on MLPclassifier with activation function = logistic with  adam algorythm
done in 2706.545s

Iteration 1, loss = 0.92503997
Iteration 2, loss = 0.31333883
Iteration 3, loss = 0.25635414
Iteration 4, loss = 0.25686164
Iteration 5, loss = 0.22814624
Iteration 6, loss = 0.20422947
Iteration 7, loss = 0.19990829
Iteration 8, loss = 0.18940044
Iteration 9, loss = 0.19696044
Iteration 10, loss = 0.17193902
Iteration 11, loss = 0.17325637
Iteration 12, loss = 0.17222052
Iteration 13, loss = 0.15904044
Iteration 14, loss = 0.17035799
Iteration 15, loss = 0.15003028
Iteration 16, loss = 0.15388962
Iteration 17, loss = 0.15095760
Iteration 18, loss = 0.15265590
Training loss did not improve more than tol=0.000100 for two consecutive epochs. Stopping.
We have  0.933214285714 of accuracy on MLPclassifier with activation function = logistic with  l-bfgs algorythm
done in 1531.047s

Iteration 1, loss = 0.49817724
Iteration 2, loss = 0.31682261
Iteration 3, loss = 0.29129329
Iteration 4, loss = 0.26827224
Iteration 5, loss = 0.27892488
Iteration 6, loss = 0.30462837
Iteration 7, loss = 0.27265325
Training loss did not improve more than tol=0.000100 for two consecutive epochs. Stopping.
We have  0.894047619048 of accuracy on MLPclassifier with activation function = tanh with  sgd algorythm
done in 558.757s

Iteration 1, loss = 0.50538451
Iteration 2, loss = 0.31496349
Iteration 3, loss = 0.30113195
Iteration 4, loss = 0.27302857
Iteration 5, loss = 0.31819666
Iteration 6, loss = 0.28583152
Iteration 7, loss = 0.27034348
Iteration 8, loss = 0.28082917
Iteration 9, loss = 0.26225840
Iteration 10, loss = 0.26584035
Iteration 11, loss = 0.26481441
Iteration 12, loss = 0.27774331
Training loss did not improve more than tol=0.000100 for two consecutive epochs. Stopping.
We have  0.921547619048 of accuracy on MLPclassifier with activation function = tanh with  adam algorythm
done in 977.087s

Iteration 1, loss = 0.46844559
Iteration 2, loss = 0.32004541
Iteration 3, loss = 0.28004486
Iteration 4, loss = 0.28970536
Iteration 5, loss = 0.28168377
Iteration 6, loss = 0.29457717
Training loss did not improve more than tol=0.000100 for two consecutive epochs. Stopping.
We have  0.892857142857 of accuracy on MLPclassifier with activation function = tanh with  l-bfgs algorythm
done in 472.654s

Iteration 1, loss = 1.29779610
Iteration 2, loss = 0.12983333
Iteration 3, loss = 0.08959002
Iteration 4, loss = 0.06646103
Iteration 5, loss = 0.05341641
Iteration 6, loss = 0.04759901
Iteration 7, loss = 0.04291960
Iteration 8, loss = 0.03530565
Iteration 9, loss = 0.04483927
Iteration 10, loss = 0.03667890
Iteration 11, loss = 0.04158188
Training loss did not improve more than tol=0.000100 for two consecutive epochs. Stopping.
We have  0.97130952381 of accuracy on MLPclassifier with activation function = relu with  sgd algorythm
done in 756.269s

Iteration 1, loss = 1.57659774
Iteration 2, loss = 0.13305658
Iteration 3, loss = 0.09183945
Iteration 4, loss = 0.06639568
Iteration 5, loss = 0.05646803
Iteration 6, loss = 0.05553462
Iteration 7, loss = 0.05456270
Iteration 8, loss = 0.03969152
Iteration 9, loss = 0.04222076
Iteration 10, loss = 0.03079309
Iteration 11, loss = 0.03401666
Iteration 12, loss = 0.02378585
Iteration 13, loss = 0.04108828
Iteration 14, loss = 0.03800991
Iteration 15, loss = 0.02471422
Training loss did not improve more than tol=0.000100 for two consecutive epochs. Stopping.
We have  0.969880952381 of accuracy on MLPclassifier with activation function = relu with  adam algorythm
done in 990.214s

Iteration 1, loss = 1.45310140
Iteration 2, loss = 0.13856403
Iteration 3, loss = 0.08906168
Iteration 4, loss = 0.07054421
Iteration 5, loss = 0.06074879
```

```
Iteration 6, loss = 0.04693585
Iteration 7, loss = 0.04625055
Iteration 8, loss = 0.03822425
Iteration 9, loss = 0.03707572
Iteration 10, loss = 0.03773546
Iteration 11, loss = 0.04171799
Iteration 12, loss = 0.03271914
Iteration 13, loss = 0.03031953
Iteration 14, loss = 0.04000638
Iteration 15, loss = 0.04103012
Iteration 16, loss = 0.02961046
Iteration 17, loss = 0.02443595
Iteration 18, loss = 0.02415082
Iteration 19, loss = 0.02096209
Iteration 20, loss = 0.03438911
Iteration 21, loss = 0.02095840
Iteration 22, loss = 0.03107915
Training loss did not improve more than tol=0.000100 for two consecutive epochs. Stopping.
We have  0.969166666667 of accuracy on MLPclassifier with activation function = relu with  l-bfgs algorythm
done in 1490.224s
```

Напоследок сверточная нейронная сеть с 5 слоями

In [80]:

```
IMAGE_SIZE = 28
NUM_CHANNELS = 1
NUM_LABELS = 10
VALIDATION_SIZE = 5000
SEED = 66478
BATCH_SIZE = 64
NUM_EPOCHS = 10
EVAL_BATCH_SIZE = 64
EVAL_FREQUENCY = 100

def error_rate(predictions, labels):
  return 100.0 - (
      100.0 *
      numpy.sum(numpy.argmax(predictions, 1) == labels) /
      predictions.shape[0])

def main(argv=None):
  train = pandas.read_csv('train.csv')
  train_data = []
  for i in range(42000):
    temp = list(convert(train[i:i + 1])[0])
    temp = np.array(temp)

    temp.shape = ((28), (28), (1))
    temp = (temp - (255 / 2.0)) / 255
    #train_data.append(list(convert(train_set[i:i + 1])[0]))
    train_data.append(temp)
  train_labels = train.label

  test = pandas.read_csv('test.csv')
  test_data = []
  for i in range(28000):
    temp = list(test[i:i + 1].iloc[:,0:].values[0])
    temp = np.array(temp)

    temp.shape = ((28), (28), (1))
    temp = (temp - (255 / 2.0)) / 255
    test_data.append(temp)

  train_data = np.array(train_data)
  test_data = np.array(test_data)
  validation_data = train_data[:VALIDATION_SIZE, ...]
  validation_labels = train_labels[:VALIDATION_SIZE]
  train_data = train_data[VALIDATION_SIZE:, ...]
  train_labels = train_labels[VALIDATION_SIZE:]
  num_epochs = NUM_EPOCHS
  train_size = train_labels.shape[0]

  train_data_node = tf.placeholder(
      tf.float32,
      shape=(BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, NUM_CHANNELS))
  train_labels_node = tf.placeholder(tf.int64, shape=(BATCH_SIZE,))
  eval_data = tf.placeholder(
      tf.float32,
      shape=(EVAL_BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, NUM_CHANNELS))

  conv1_weights = tf.Variable(
      tf.truncated_normal([5, 5, NUM_CHANNELS, 32],
                          stddev=0.1,
                          seed=SEED))
  conv1_biases = tf.Variable(tf.zeros([32]))
  conv2_weights = tf.Variable(
      tf.truncated_normal([5, 5, 32, 64],
                          stddev=0.1,
                          seed=SEED))
  conv2_biases = tf.Variable(tf.constant(0.1, shape=[64]))
  fc1_weights = tf.Variable(
      tf.truncated_normal(
          [IMAGE_SIZE // 4 * IMAGE_SIZE // 4 * 64, 512],
          stddev=0.1,
          seed=SEED))
  fc1_biases = tf.Variable(tf.constant(0.1, shape=[512]))
  fc2_weights = tf.Variable(
      tf.truncated_normal([512, NUM_LABELS],
                          stddev=0.1,
                          seed=SEED))
  fc2_biases = tf.Variable(tf.constant(0.1, shape=[NUM_LABELS]))

  def model(data, train=False):
    conv = tf.nn.conv2d(data,
                        conv1_weights,
                        strides=[1, 1, 1, 1],
                        padding='SAME')
```

```
                          padding='SAME')
    relu = tf.nn.relu(tf.nn.bias_add(conv, conv1_biases))

    pool = tf.nn.max_pool(relu,
                          ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1],
                          padding='SAME')
    conv = tf.nn.conv2d(pool,
                        conv2_weights,
                        strides=[1, 1, 1, 1],
                        padding='SAME')
    relu = tf.nn.relu(tf.nn.bias_add(conv, conv2_biases))
    pool = tf.nn.max_pool(relu,
                          ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1],
                          padding='SAME')
    pool_shape = pool.get_shape().as_list()
    reshape = tf.reshape(
        pool,
        [pool_shape[0], pool_shape[1] * pool_shape[2] * pool_shape[3]])
    hidden = tf.nn.relu(tf.matmul(reshape, fc1_weights) + fc1_biases)

    if train:
      hidden = tf.nn.dropout(hidden, 0.5, seed=SEED)
    return tf.matmul(hidden, fc2_weights) + fc2_biases

  logits = model(train_data_node, True)
  loss = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
      logits, train_labels_node))

  regularizers = (tf.nn.l2_loss(fc1_weights) + tf.nn.l2_loss(fc1_biases) +
                  tf.nn.l2_loss(fc2_weights) + tf.nn.l2_loss(fc2_biases))
  loss += 5e-4 * regularizers

  batch = tf.Variable(0)
  learning_rate = tf.train.exponential_decay(
      0.01,
      batch * BATCH_SIZE,
      train_size,
      0.95,
      staircase=True)

  optimizer = tf.train.MomentumOptimizer(learning_rate,
                                         0.9).minimize(loss,
                                                       global_step=batch)

  train_prediction = tf.nn.softmax(logits)

  eval_prediction = tf.nn.softmax(model(eval_data))

  def eval_in_batches(data, sess):
    size = data.shape[0]
    if size < EVAL_BATCH_SIZE:
      raise ValueError("batch size for evals larger than dataset: %d" % size)
    predictions = numpy.ndarray(shape=(size, NUM_LABELS), dtype=numpy.float32)
    for begin in xrange(0, size, EVAL_BATCH_SIZE):
      end = begin + EVAL_BATCH_SIZE
      if end <= size:
        predictions[begin:end, :] = sess.run(
            eval_prediction,
            feed_dict={eval_data: data[begin:end, ...]})
      else:
        batch_predictions = sess.run(
            eval_prediction,
            feed_dict={eval_data: data[-EVAL_BATCH_SIZE:, ...]})
        predictions[begin:, :] = batch_predictions[begin - size:, :]
    return predictions

  start_time = time.time()
  with tf.Session() as sess:
    tf.initialize_all_variables().run()
    print('Initialized!')
    for step in xrange(int(num_epochs * train_size) // BATCH_SIZE):
      offset = (step * BATCH_SIZE) % (train_size - BATCH_SIZE)
      batch_data = train_data[offset:(offset + BATCH_SIZE), ...]
      batch_labels = train_labels[offset:(offset + BATCH_SIZE)]
      feed_dict = {train_data_node: batch_data,
                   train_labels_node: batch_labels}

      _, l, lr, predictions = sess.run(
          [optimizer, loss, learning_rate, train_prediction],
          feed_dict=feed_dict)
      if step % EVAL_FREQUENCY == 0:
        elapsed_time = time.time() - start_time
        start_time = time.time()
        print('Step %d (epoch %.2f), %.1f ms' %
              (step, float(step) * BATCH_SIZE / train_size,
               1000 * elapsed_time / EVAL_FREQUENCY))
        print('Minibatch loss: %.3f, learning rate: %.6f' % (l, lr))
        print('Minibatch error: %.1f%%' % error_rate(predictions, batch_labels))
        print('Validation error: %.1f%%' % error_rate(
            eval_in_batches(validation_data, sess), validation_labels))
        sys.stdout.flush()

    print('Final!')
    out = eval_in_batches(test_data, sess)
    pandas.DataFrame(out).to_csv('Kaggle.csv')

if __name__ == '__main__':
  tf.app.run()
```

```
Initialized!
Step 0 (epoch 0.00), 7.7 ms
Minibatch loss: 11.511, learning rate: 0.010000
Minibatch error: 96.9%
Validation error: 84.3%
Step 100 (epoch 0.17), 112.2 ms
Minibatch loss: 3.473, learning rate: 0.010000
Minibatch error: 18.8%
Validation error: 6.6%
```

```
Step 200 (epoch 0.35), 115.0 ms
Minibatch loss: 3.358, learning rate: 0.010000
Minibatch error: 10.9%
Validation error: 4.2%
Step 300 (epoch 0.52), 120.5 ms
Minibatch loss: 3.383, learning rate: 0.010000
Minibatch error: 7.8%
Validation error: 3.3%
Step 400 (epoch 0.69), 119.0 ms
Minibatch loss: 3.109, learning rate: 0.010000
Minibatch error: 4.7%
Validation error: 2.6%
Step 500 (epoch 0.86), 117.9 ms
Minibatch loss: 3.120, learning rate: 0.010000
Minibatch error: 6.2%
Validation error: 2.4%
Step 600 (epoch 1.04), 114.5 ms
Minibatch loss: 3.098, learning rate: 0.009500
Minibatch error: 4.7%
Validation error: 2.1%
Step 700 (epoch 1.21), 113.6 ms
Minibatch loss: 2.999, learning rate: 0.009500
Minibatch error: 3.1%
Validation error: 2.5%
Step 800 (epoch 1.38), 111.6 ms
Minibatch loss: 2.948, learning rate: 0.009500
Minibatch error: 3.1%
Validation error: 2.1%
Step 900 (epoch 1.56), 115.9 ms
Minibatch loss: 2.925, learning rate: 0.009500
Minibatch error: 3.1%
Validation error: 1.6%
Step 1000 (epoch 1.73), 117.1 ms
Minibatch loss: 2.901, learning rate: 0.009500
Minibatch error: 3.1%
Validation error: 1.8%
Step 1100 (epoch 1.90), 121.1 ms
Minibatch loss: 2.846, learning rate: 0.009500
Minibatch error: 0.0%
Validation error: 1.8%
Step 1200 (epoch 2.08), 113.3 ms
Minibatch loss: 2.821, learning rate: 0.009025
Minibatch error: 1.6%
Validation error: 1.9%
Step 1300 (epoch 2.25), 116.5 ms
Minibatch loss: 2.842, learning rate: 0.009025
Minibatch error: 4.7%
Validation error: 1.7%
Step 1400 (epoch 2.42), 117.7 ms
Minibatch loss: 2.742, learning rate: 0.009025
Minibatch error: 0.0%
Validation error: 1.5%
Step 1500 (epoch 2.59), 111.5 ms
Minibatch loss: 2.752, learning rate: 0.009025
Minibatch error: 1.6%
Validation error: 1.6%
Step 1600 (epoch 2.77), 110.0 ms
Minibatch loss: 2.706, learning rate: 0.009025
Minibatch error: 0.0%
Validation error: 1.5%
Step 1700 (epoch 2.94), 120.2 ms
Minibatch loss: 2.766, learning rate: 0.009025
Minibatch error: 1.6%
Validation error: 1.5%
Step 1800 (epoch 3.11), 112.3 ms
Minibatch loss: 2.683, learning rate: 0.008574
Minibatch error: 1.6%
Validation error: 1.6%
Step 1900 (epoch 3.29), 111.0 ms
Minibatch loss: 2.670, learning rate: 0.008574
Minibatch error: 1.6%
Validation error: 1.5%
Step 2000 (epoch 3.46), 114.4 ms
Minibatch loss: 2.636, learning rate: 0.008574
Minibatch error: 1.6%
Validation error: 1.2%
Step 2100 (epoch 3.63), 110.3 ms
Minibatch loss: 2.577, learning rate: 0.008574
Minibatch error: 0.0%
Validation error: 1.3%
Step 2200 (epoch 3.81), 110.3 ms
Minibatch loss: 2.576, learning rate: 0.008574
Minibatch error: 1.6%
Validation error: 1.3%
Step 2300 (epoch 3.98), 112.5 ms
Minibatch loss: 2.627, learning rate: 0.008574
Minibatch error: 3.1%
Validation error: 1.3%
Step 2400 (epoch 4.15), 110.9 ms
Minibatch loss: 2.517, learning rate: 0.008145
Minibatch error: 0.0%
Validation error: 1.2%
Step 2500 (epoch 4.32), 109.9 ms
Minibatch loss: 2.509, learning rate: 0.008145
Minibatch error: 0.0%
Validation error: 1.3%
Step 2600 (epoch 4.50), 110.6 ms
Minibatch loss: 2.504, learning rate: 0.008145
Minibatch error: 3.1%
Validation error: 1.2%
Step 2700 (epoch 4.67), 110.2 ms
Minibatch loss: 2.547, learning rate: 0.008145
Minibatch error: 1.6%
Validation error: 1.2%
Step 2800 (epoch 4.84), 110.4 ms
Minibatch loss: 2.441, learning rate: 0.008145
Minibatch error: 0.0%
Validation error: 1.2%
Step 2900 (epoch 5.02), 110.3 ms
Minibatch loss: 2.414, learning rate: 0.007738
```

```
Minibatch error: 0.0%
Validation error: 1.2%
Step 3000 (epoch 5.19), 110.4 ms
Minibatch loss: 2.427, learning rate: 0.007738
Minibatch error: 3.1%
Validation error: 1.1%
Step 3100 (epoch 5.36), 110.7 ms
Minibatch loss: 2.393, learning rate: 0.007738
Minibatch error: 0.0%
Validation error: 1.2%
Step 3200 (epoch 5.54), 110.1 ms
Minibatch loss: 2.407, learning rate: 0.007738
Minibatch error: 1.6%
Validation error: 1.2%
Step 3300 (epoch 5.71), 116.8 ms
Minibatch loss: 2.435, learning rate: 0.007738
Minibatch error: 1.6%
Validation error: 1.1%
Step 3400 (epoch 5.88), 122.9 ms
Minibatch loss: 2.321, learning rate: 0.007738
Minibatch error: 0.0%
Validation error: 1.2%
Step 3500 (epoch 6.05), 117.7 ms
Minibatch loss: 2.356, learning rate: 0.007351
Minibatch error: 3.1%
Validation error: 0.9%
Step 3600 (epoch 6.23), 124.7 ms
Minibatch loss: 2.289, learning rate: 0.007351
Minibatch error: 0.0%
Validation error: 1.1%
Step 3700 (epoch 6.40), 111.2 ms
Minibatch loss: 2.331, learning rate: 0.007351
Minibatch error: 3.1%
Validation error: 1.0%
Step 3800 (epoch 6.57), 123.1 ms
Minibatch loss: 2.256, learning rate: 0.007351
Minibatch error: 0.0%
Validation error: 1.1%
Step 3900 (epoch 6.75), 111.1 ms
Minibatch loss: 2.236, learning rate: 0.007351
Minibatch error: 0.0%
Validation error: 1.1%
Step 4000 (epoch 6.92), 113.5 ms
Minibatch loss: 2.219, learning rate: 0.007351
Minibatch error: 0.0%
Validation error: 1.1%
Step 4100 (epoch 7.09), 110.8 ms
Minibatch loss: 2.253, learning rate: 0.006983
Minibatch error: 1.6%
Validation error: 1.1%
Step 4200 (epoch 7.26), 110.5 ms
Minibatch loss: 2.219, learning rate: 0.006983
Minibatch error: 0.0%
Validation error: 1.0%
Step 4300 (epoch 7.44), 110.2 ms
Minibatch loss: 2.245, learning rate: 0.006983
Minibatch error: 3.1%
Validation error: 1.0%
Step 4400 (epoch 7.61), 117.3 ms
Minibatch loss: 2.175, learning rate: 0.006983
Minibatch error: 0.0%
Validation error: 0.9%
Step 4500 (epoch 7.78), 131.0 ms
Minibatch loss: 2.171, learning rate: 0.006983
Minibatch error: 1.6%
Validation error: 1.0%
Step 4600 (epoch 7.96), 114.2 ms
Minibatch loss: 2.186, learning rate: 0.006983
Minibatch error: 1.6%
Validation error: 1.0%
Step 4700 (epoch 8.13), 120.0 ms
Minibatch loss: 2.145, learning rate: 0.006634
Minibatch error: 1.6%
Validation error: 1.0%
Step 4800 (epoch 8.30), 112.3 ms
Minibatch loss: 2.100, learning rate: 0.006634
Minibatch error: 0.0%
Validation error: 1.0%
Step 4900 (epoch 8.48), 110.5 ms
Minibatch loss: 2.119, learning rate: 0.006634
Minibatch error: 1.6%
Validation error: 1.0%
Step 5000 (epoch 8.65), 114.2 ms
Minibatch loss: 2.096, learning rate: 0.006634
Minibatch error: 1.6%
Validation error: 0.9%
Step 5100 (epoch 8.82), 114.4 ms
Minibatch loss: 2.061, learning rate: 0.006634
Minibatch error: 0.0%
Validation error: 1.0%
Step 5200 (epoch 8.99), 135.1 ms
Minibatch loss: 2.064, learning rate: 0.006634
Minibatch error: 0.0%
Validation error: 1.0%
Step 5300 (epoch 9.17), 146.3 ms
Minibatch loss: 2.044, learning rate: 0.006302
Minibatch error: 0.0%
Validation error: 1.0%
Step 5400 (epoch 9.34), 124.3 ms
Minibatch loss: 2.042, learning rate: 0.006302
Minibatch error: 1.6%
Validation error: 0.9%
Step 5500 (epoch 9.51), 115.9 ms
Minibatch loss: 2.049, learning rate: 0.006302
Minibatch error: 3.1%
Validation error: 1.0%
Step 5600 (epoch 9.69), 116.6 ms
Minibatch loss: 1.994, learning rate: 0.006302
Minibatch error: 0.0%
Validation error: 0.8%
```

```
Step 5700 (epoch 9.86), 115.9 ms
Minibatch loss: 1.984, learning rate: 0.006302
Minibatch error: 0.0%
Validation error: 0.9%
Final!
```

```
An exception has occurred, use %tb to see the full traceback.

SystemExit
To exit: use 'exit', 'quit', or Ctrl-D.
```

In [ ]:

```
test = pandas.read_csv('Kaggle.csv')
pred = []
for i in range(28000):
    temp = list(test[i:i + 1].iloc[:,1:].values[0])
    mn = max(temp)
    for i in range(len(temp)):
        if mn == temp[i]:
            pred.append(i)
            break
sample = pandas.read_csv('sample_submission.csv')
kaggle_sub(sample,pred)
```

Сабмит на Kaggle, имеем 144 место в общем зачете и score=0.99171.

Третье домашнее задание

In [54]:

```
import pandas
import matplotlib as plt
%matplotlib inline
import csv
import seaborn as sls
from scipy.stats import kendalltau
from time import time
from sklearn.ensemble import ExtraTreesClassifier
from sklearn import cross_validation
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import cv2
import numpy as np
from pylab import *
from scipy.spatial import distance
from scipy.ndimage import interpolation
import tables
import random as pyrandom
from sklearn import neighbors
def log_progress(sequence, every=None, size=None):
    from ipywidgets import IntProgress, HTML, VBox
    from IPython.display import display

    is_iterator = False
    if size is None:
        try:
            size = len(sequence)
        except TypeError:
            is_iterator = True
    if size is not None:
        if every is None:
            if size <= 200:
                every = 1
            else:
                every = size / 200     # every 0.5%
    else:
        assert every is not None, 'sequence is iterator, set every'

    if is_iterator:
        progress = IntProgress(min=0, max=1, value=1)
        progress.bar_style = 'info'
    else:
        progress = IntProgress(min=0, max=size, value=0)
    label = HTML()
    box = VBox(children=[label, progress])
    display(box)

    index = 0
    try:
        for index, record in enumerate(sequence, 1):
            if index == 1 or index % every == 0:
                if is_iterator:
                    label.value = '{index} / ?'.format(index=index)
                else:
                    progress.value = index
                    label.value = u'{index} / {size}'.format(
                        index=index,
                        size=size
                    )
            yield record
    except:
        progress.bar_style = 'danger'
        raise
    else:
        progress.bar_style = 'success'
        progress.value = index
        label.value = str(index or '?')
```

In [26]:

```
train_set = pandas.read_csv("train.csv")
x = train_set.ix[: , 1:]
x = np.array(x, dtype = 'float64')
```

In [ ]:

```
images_impoved = []
for i in range(41999):
    elem = np.array(data[i:i+1])[0]
    temp = []
    for el in elem:
        for ele in el:
            temp.append(ele)
    images_impoved.append(temp)
```

In [27]:

```
def convert(frame):
    return frame.iloc[:,1:].values

def show_pic(img):
    img = img.reshape(28,28)
    #plt.pyplot.axis('off')
    plt.pyplot.imshow(img, interpolation='nearest')

def save_pic(img):
    #plt.pyplot.savefig('img.jpg', bbox_inches='tight')
    plt.pyplot.savefig('img.jpg')

def deskew (image):
    affine_flags = cv2.WARP_INVERSE_MAP | cv2.INTER_LINEAR
    size = len(image)
    m = cv2.moments(image)

    if abs(m['mu02']) < 1e2:
        return image.copy()

    skew = m['mu11'] / m['mu02']
    M = np.float32([[1, skew, 0.5 * size * skew], [0 , 1 , 0]])
    image = cv2.warpAffine(image , M , (size , size) , flags = affine_flags)
    return image
```

Выравнивание

In [28]:

```
i = 5
plt.axis('off')
plt.imshow(x[i].reshape(28,28))

plt.figure()
plt.axis('off')
plt.imshow(deskew(x[i].reshape(28,28)),)
```

Out[28]:

```
<matplotlib.image.AxesImage at 0x1aa6bd1c358>
```

In [ ]:

```
images = []
skew = []

for row in log_progress(x):
    image = row.reshape(28 , 28)
    skewed_image = deskew(image)
```

```
    images.append(skewed_image)
    skew.append(skewed_image.flatten())
    x = np.array(skew, dtype = 'float64' )
```

In [ ]:

```
images_impoved = []
for elem in images:
    temp = []
    for el in elem.flat:
        temp.append(el)
    images_impoved.append(temp)
```

In [ ]:

```
csv.writer(open('images.csv', 'w'), dialect='excel').writerows(images)
```

In [29]:

```
y = train_set.label

X_train, X_test, y_train, y_test = cross_validation.train_test_split(images_improved, y, test_size=0.2, random_state=0)
```

In [35]:

```
n_jobs = -1

print("Started to fitting KNN on data with %d cores..." % 8 if n_jobs == -1 else n_jobs)

accuracy_values = []
for number in range(1,3):
    t0 = time()

    clf = neighbors.KNeighborsClassifier(number, n_jobs=n_jobs, weights='uniform')

    clf.fit(X_train, y_train)

    y_predict = clf.predict(X_test)

    temp_acc = accuracy_score(y_test, y_predict)
    print('We have ', temp_acc, 'of accuracy on KNN with ', number, ' neighbor')
    accuracy_values.append(temp_acc)
    print("done in %0.3fs" % (time() - t0))
    print()
```

```
Started to fitting KNN on data with 8 cores...
We have  0.970952380952 of accuracy on KNN with  1  neighbor
done in 88.428s

We have  0.964404761905 of accuracy on KNN with  2  neighbor
done in 88.318s
```

In [36]:

```
print("Started to fitting poly SVM on data")

accuracy_values_poly_SVM = []
for number in range(2,4):
    t0 = time()

    clf = SVC(kernel='poly', degree=number)

    clf.fit(X_train, y_train)

    y_predict = clf.predict(X_test)

    temp_acc = accuracy_score(y_test, y_predict)
    print('We have ', temp_acc, 'of accuracy on poly SVC with degree =', number)
    accuracy_values_poly_SVM.append(temp_acc)
    print("done in %0.3fs" % (time() - t0))
    print()
```

```
Started to fitting poly SVM on data
We have  0.977738095238 of accuracy on poly SVC with degree = 2
done in 176.522s

We have  0.974047619048 of accuracy on poly SVC with degree = 3
done in 175.887s
```

In [37]:

```
n_jobs = -1

print("Started to fitting ExtraTreesClassifier on data with %d cores..." % 8 if n_jobs == -1 else n_jobs)

t0 = time()

forest = ExtraTreesClassifier(n_estimators=1000,
                              max_features=128,
                              n_jobs=n_jobs,
                              random_state=0)

forest.fit(X_train, y_train)

y_predict = forest.predict(X_test)

print('we have ', accuracy_score(y_test, y_predict), 'of accuracy on ExtraTreeClassifier')

print("done in %0.3fs" % (time() - t0))
```

```
Started to fitting ExtraTreesClassifier on data with 8 cores...
we have  0.97119047619 of accuracy on ExtraTreeClassifier
done in 249.393s
```

Никакого прироста в точности =(

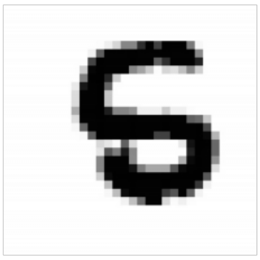
Эрозия и дилатация

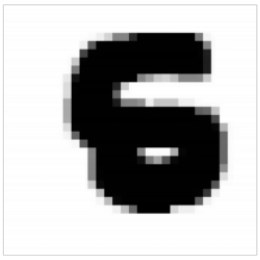
In [39]:

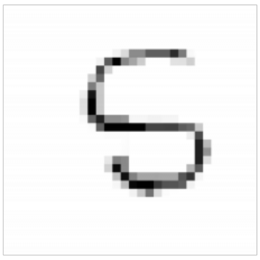
```
img = cv2.imread('img.jpg', 0)

kernel = np.ones((23,23), np.uint8)

img_erosion = cv2.erode(img, kernel, iterations=1)
img_dilation = cv2.dilate(img, kernel, iterations=1)

plt.axis('off')

plt.imshow(img , cmap = 'gray', interpolation = 'bicubic')
plt.figure()
plt.axis('off')
plt.imshow(img_erosion, cmap = 'gray', interpolation = 'bicubic')
plt.figure()
plt.axis('off')
plt.imshow(img_dilation, cmap = 'gray', interpolation = 'bicubic')
```

Out[39]:

<matplotlib.image.AxesImage at 0x1aa2362dbe0>







In [ ]:

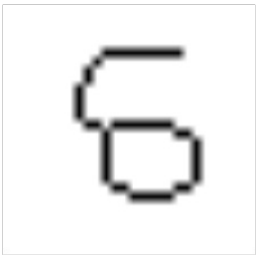
```
Скелет алгоритмом Розенфильда
```

In [46]:

```
from skimage.morphology import skeletonize
from skimage import draw

image = x[19]

for i in range(len(image)):
    if image[i] > 0:
        image[i] = 1

skeleton = skeletonize(image.reshape(28,28))
plt.axis('off')
plt.imshow(skeleton)
```

Out[46]:

<matplotlib.image.AxesImage at 0x1aa1f74aeb8>



Попробуем добавить новые фичи

In [50]:

```
train_set = pandas.read_csv("train.csv")
x = train_set.ix[: , 1:]
x = list(np.array(x, dtype = 'float64'))
y = train_set.label
```

Простая статистика количества заполненных\пустых клеток и их отношение

In [58]:

```
x = np.array(x)
x_ls = []
for i in range(len(x)):
    temp = []
    white = 0
    black = 0
    for elem in x[i]:
        temp.append(elem)
        if elem > 0:
            black += 1
        else:
            white += 1
    temp.append(white)
    temp.append(black)
    temp.append(white/black)
    x_ls.append(temp)
```

In [66]:

```
X_train, X_test, y_train, y_test = cross_validation.train_test_split(x_ls, y, test_size=0.2, random_state=0)
```

In [67]:

```
n_jobs = -1

print("Started to fitting ExtraTreesClassifier on data with %d cores..." % 8 if n_jobs == -1 else n_jobs)

t0 = time()

forest = ExtraTreesClassifier(n_estimators=500,
                              max_features=128,
                              n_jobs=n_jobs,
                              random_state=0)


forest.fit(X_train, y_train)

y_predict = forest.predict(X_test)

print('we have ', accuracy_score(y_test, y_predict), 'of accuracy on ExtraTreeClassifier')

print("done in %0.3fs" % (time() - t0))

importances = forest.feature_importances_
```

```
Started to fitting ExtraTreesClassifier on data with 8 cores...
we have  0.970714285714 of accuracy on ExtraTreeClassifier
done in 121.077s
```

In [69]:

```
indices = np.argsort(importances)[::-1]

print("Feature ranking:")

for f in range(np.array(x_ls).shape[1]):
    if indices[f] >= 784:
        print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))
```

```
Feature ranking:
3. feature 786 (0.011099)
6. feature 784 (0.009398)
12. feature 785 (0.007731)
```

Получаем, что данные характеристики очень важны при классификации.

Четвертое домашнее задание

In [86]:

```
import pandas
import numpy as np
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
import csv
import sklearn.cross_validation
from time import time
from sklearn import neighbors
from sklearn.metrics import accuracy_score
from sklearn.ensemble import ExtraTreesClassifier
import sklearn.manifold.mds as mds
from sklearn.decomposition import PCA
from matplotlib.patches import FancyArrowPatch
from mpl_toolkits.mplot3d import proj3d
from sklearn.decomposition import KernelPCA
from IPython.display import Image
from IPython.core.display import HTML
```

```
def log_progress(sequence, every=None, size=None):
    from ipywidgets import IntProgress, HTML, VBox
    from IPython.display import display

    is_iterator = False
    if size is None:
        try:
            size = len(sequence)
        except TypeError:
            is_iterator = True
    if size is not None:
        if every is None:
            if size <= 200:
                every = 1
            else:
                every = size / 200     # every 0.5%
    else:
        assert every is not None, 'sequence is iterator, set every'

    if is_iterator:
        progress = IntProgress(min=0, max=1, value=1)
        progress.bar_style = 'info'
    else:
        progress = IntProgress(min=0, max=size, value=0)
    label = HTML()
    box = VBox(children=[label, progress])
    display(box)

    index = 0
    try:
        for index, record in enumerate(sequence, 1):
            if index == 1 or index % every == 0:
                if is_iterator:
                    label.value = '{index} / ?'.format(index=index)
                else:
                    progress.value = index
                    label.value = u'{index} / {size}'.format(
                        index=index,
                        size=size
                    )
            yield record
    except:
        progress.bar_style = 'danger'
        raise
    else:
        progress.bar_style = 'success'
        progress.value = index
        label.value = str(index or '?')
```

In [2]:

```
train_set = pandas.read_csv('train.csv')
```

In [3]:

```
x = train_set.ix[: , 1:]
x = np.array(x, dtype = 'float64')
```

In [95]:

```
clf = PCA(n_components=9)
clf.fit(x)
```

Out[95]:

```
PCA(copy=True, n_components=9, whiten=False)
```

In [23]:

```
cov_mat = np.array(clf.get_covariance())
```

Важность признаков

In [6]:

```
print(clf.explained_variance_ratio_)
x_trans = clf.transform(x)
```

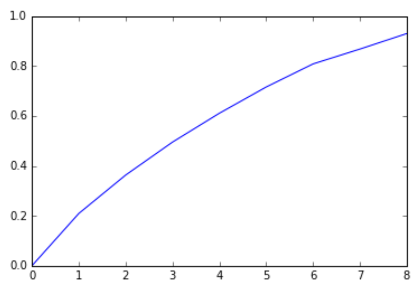
```
[ 0.09748938  0.07160266  0.06145903  0.05379302  0.04894262  0.04303214
  0.03277051  0.02892103  0.02766902]
```

In [11]:

```
evals, evecs = np.linalg.eig(np.cov(x_trans.transpose()))

n_features_plot = [sum(sorted(evals[:i])) / sum(evals) for i in reversed(range(len(evals)))]

plt.plot([i for i in reversed(range(9))], n_features_plot)
```

Out[11]:

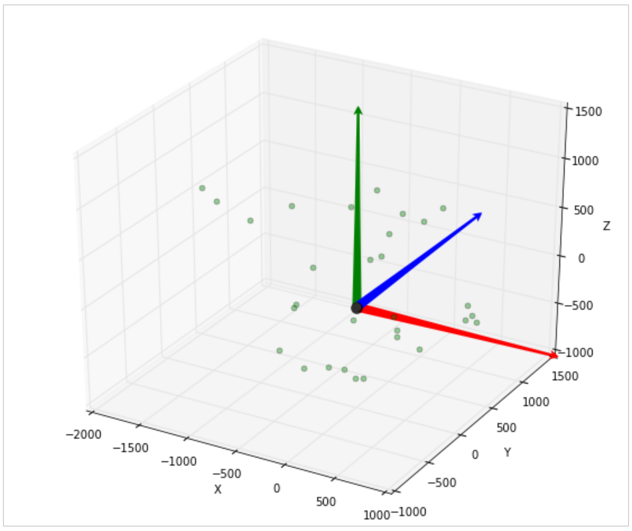
```
[<matplotlib.lines.Line2D at 0x13d0045f0f0>]
```



Собственные трехмерные векторы

In [68]:

```
class Arrow3D(FancyArrowPatch):
    def __init__(self, xs, ys, zs, *args, **kwargs):
        FancyArrowPatch.__init__(self, (0,0), (0,0), *args, **kwargs)
        self._verts3d = xs, ys, zs

    def draw(self, renderer):
        xs3d, ys3d, zs3d = self._verts3d
        xs, ys, zs = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M)
        self.set_positions((xs[0],ys[0]),(xs[1],ys[1]))
        FancyArrowPatch.draw(self, renderer)

fig = plt.figure(figsize=(10,10))
ax = fig.gca(projection='3d')
ax.set_aspect("equal")

clf = PCA(n_components=3)
clf.fit(x)
x_trans = clf.transform(x)
evals, evecs = np.linalg.eig(np.cov(x_trans.transpose()))

mean_x = np.mean(x_trans[:,0])
mean_y = np.mean(x_trans[:,1])
mean_z = np.mean(x_trans[:,2])

ax.plot(x_trans[:,0][:30], x_trans[:,1][:30], x_trans[:,2][:30], 'o', markersize=5, color='g', alpha=0.4)
ax.plot([mean_x], [mean_y], [mean_z], 'o', markersize=10, color='black', alpha=0.8)

a = Arrow3D([mean_x, evecs[0][0] * 2000], [mean_y, evecs[0][1] * 2000], [mean_z, evecs[0][2] * 2000],
            mutation_scale=20, lw=0.05, arrowstyle="fancy", color="r")
ax.add_artist(a)
ax.set_xlabel('X')

b = Arrow3D([mean_x, evecs[1][0] * 2000], [mean_y, evecs[1][1] * 2000], [mean_z, evecs[1][2] * 2000],
            mutation_scale=20, lw=0.05, arrowstyle="fancy", color="g")
ax.add_artist(b)
ax.set_ylabel('Y')

c = Arrow3D([mean_x, evecs[2][0] * 2000], [mean_y, evecs[2][1] * 2000], [mean_z, evecs[2][2] * 2000],
            mutation_scale=20, lw=0.05, arrowstyle="fancy", color="b")
ax.add_artist(c)
ax.set_zlabel('Z')

plt.show()
```



In [96]:

```
trans_data = clf.transform(x)
```

In [25]:

```
trans_data[1]
```

Out[25]:

```
array([-1701.4516848 ,   360.5515562 ,   501.80559391,   335.42365557,
        -442.37893255,   738.40404869,   653.87543763,  -176.60067741,
           7.52017489])
```

In [26]:

```
minpos = min(cov_mat.flat)
```

In [27]:

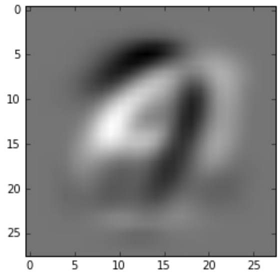
```
data_flat = []
for elem in cov_mat:
    temp = []
    for el in elem:
        temp.append(el + minpos)
    data_flat.append(temp)
```

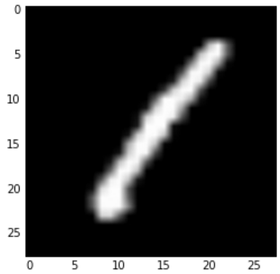
Картинки для различных степеней сжатия

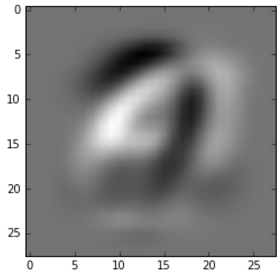
In [105]:

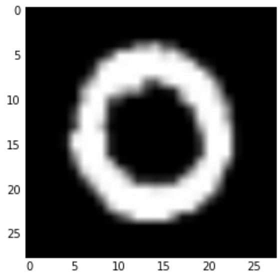
```
restored_data = clf.inverse_transform(trans_data)
```

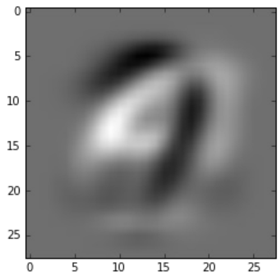
```python
for i in range(10):
    plt.figure()
    for j in range(len(trans_data[i])):
        trans_data[i][j] += min(trans_data[i])
    plt.imshow(restored_data[i].reshape(28,28), cmap='gray')
    plt.figure()
    plt.imshow(x[i].reshape(28,28), cmap='gray')
```

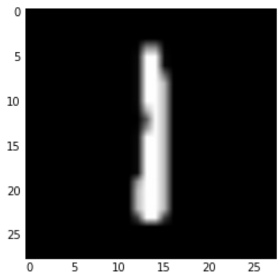


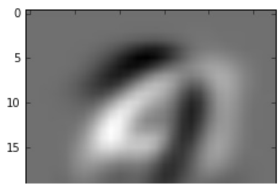


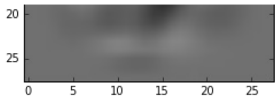


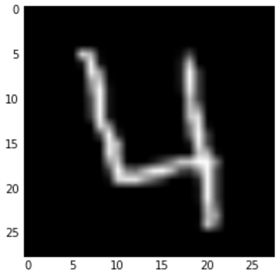


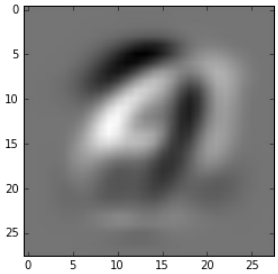


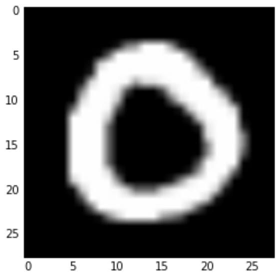


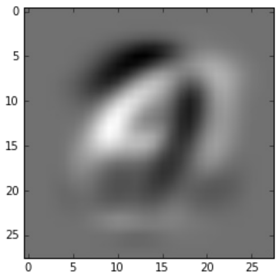


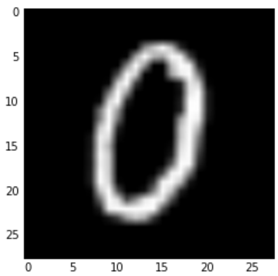

20
25
0 5 10 15 20 25

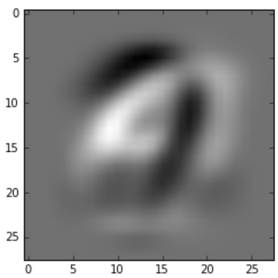

0
5
10
15
20
25
0 5 10 15 20 25


0
5
10
15
20
25
0 5 10 15 20 25

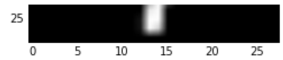

0
5
10
15
20
25
0 5 10 15 20 25

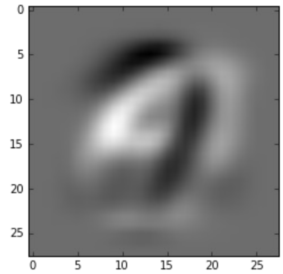

0
5
10
15
20
25
0 5 10 15 20 25

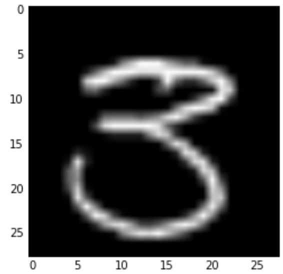

0
5
10
15
20
25
0 5 10 15 20 25

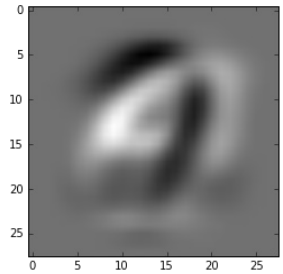

0
5
10
15
20
25
0 5 10 15 20 25

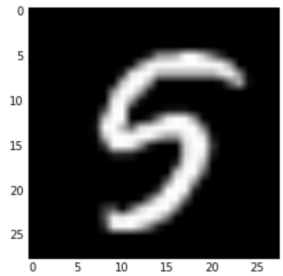

0
5
10
15
20

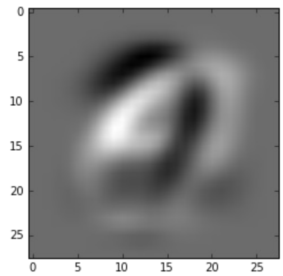


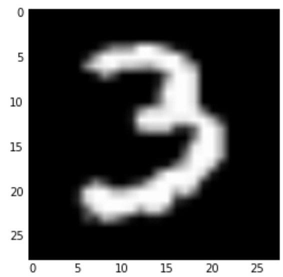








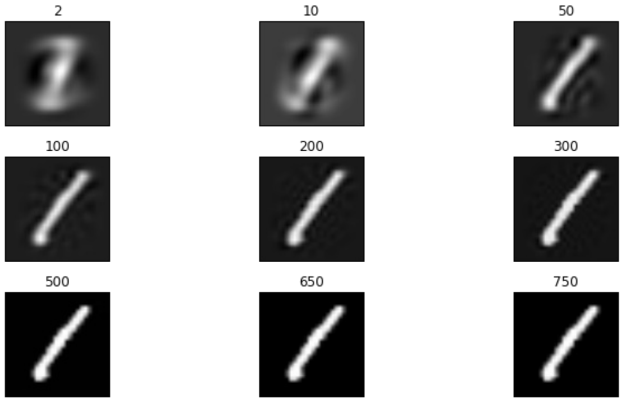




In [109]:

```
images = []
pca_number = [2, 10, 50, 100, 200, 300, 500, 650, 750]
for number in pca_number:
    clf = PCA(n_components=number)
    clf.fit(x)
    trans_data = clf.transform(x)

    restored_data = clf.inverse_transform(trans_data)

    images.append(restored_data[0])
```

```
fig, axes = plt.subplots(3, 3, figsize=(12, 6),
                         subplot_kw={'xticks': [], 'yticks': []})

fig.subplots_adjust(hspace=0.3, wspace=0.05)

i = 0
for ax, interp_method in zip(axes.flat, pca_number):
    ax.imshow(images[i].reshape(28,28), cmap='gray')
    ax.set_title(interp_method)
    i += 1
plt.show()
```



Поссмотрим на показания классификаторов

In [70]:

```
y = train_set['label']
```

In [71]:

```
clf = PCA(n_components=9)
clf.fit(x)
trans_data = clf.transform(x)
```

In [72]:

```
X_train, X_test, y_train, y_test = sklearn.cross_validation.train_test_split(trans_data, y, test_size=0.2, random_state=0)
```

In [73]:

```
n_jobs = -1

print("Started to fitting KNN on data with %d cores..." % 8 if n_jobs == -1 else n_jobs)

accuracy_values = []
for number in range(1, 10):
    t0 = time()

    clf = neighbors.KNeighborsClassifier(number, n_jobs=n_jobs, weights='uniform')

    clf.fit(X_train, y_train)

    y_predict = clf.predict(X_test)

    temp_acc = accuracy_score(y_test, y_predict)
    print('We have ', temp_acc, 'of accuracy on KNN with ', number, ' neighbor')
    accuracy_values.append(temp_acc)
    print("done in %0.3fs" % (time() - t0))
    print()
```

```
Started to fitting KNN on data with 8 cores...
We have  0.89880952381 of accuracy on KNN with  1  neighbor
done in 0.285s

We have  0.893928571429 of accuracy on KNN with  2  neighbor
done in 0.268s

We have  0.908928571429 of accuracy on KNN with  3  neighbor
done in 0.410s

We have  0.911071428571 of accuracy on KNN with  4  neighbor
done in 0.380s

We have  0.9125 of accuracy on KNN with  5  neighbor
done in 0.396s

We have  0.914642857143 of accuracy on KNN with  6  neighbor
done in 0.425s

We have  0.91369047619 of accuracy on KNN with  7  neighbor
done in 0.392s

We have  0.916666666667 of accuracy on KNN with  8  neighbor
done in 0.561s

We have  0.914880952381 of accuracy on KNN with  9  neighbor
done in 0.528s
```

Очень приятное быстродействие при умеренном снижении качества

In [75]:

```
n_jobs = -1

print("Started to fitting ExtraTreesClassifier on data with %d cores..." % 8 if n_jobs == -1 else n_jobs)
```

```
t0 = time()

forest = ExtraTreesClassifier(n_estimators=2000,
                              max_features=9,
                              n_jobs=n_jobs,
                              random_state=0)

forest.fit(X_train, y_train)

y_predict = forest.predict(X_test)

print('we have ', accuracy_score(y_test, y_predict), 'of accuracy on ExtraTreeClassifier')

print("done in %0.3fs" % (time() - t0))
```

```
Started to fitting ExtraTreesClassifier on data with 8 cores...
we have  0.908214285714 of accuracy on ExtraTreeClassifier
done in 25.417s
```

Попробуем нарисовать распределение

In [110]:

```
clf = PCA(n_components=2)
clf.fit(x)
```

Out[110]:

```
PCA(copy=True, n_components=2, whiten=False)
```

In [115]:

```
plot_transform = clf.transform(x)
```

In [119]:

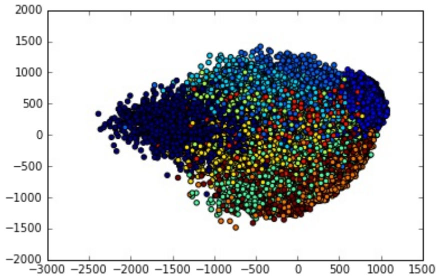
```
plot_transform = np.array(plot_transform)
#plt.scatter(plot_transform[])
```

In [121]:

```
scat_x = []
scat_y = []
for elem in plot_transform:
    scat_x.append(elem[0])
    scat_y.append(elem[1])
```

In [129]:

```
img = plt.scatter(scat_x,scat_y, c=y, alpha=1)
plt.savefig('2dplot.jpg')
```



In [78]:

```
clf = KernelPCA(n_components=10, kernel='poly', degree=2)
clf.fit(x)
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-78-239529c0dbcc> in <module>()
      1 clf = KernelPCA(n_components=10, kernel='poly', degree=2)
----> 2 clf.fit(x)

C:\Users\gamer\Anaconda3\lib\site-packages\sklearn\decomposition\kernel_pca.py in fit(self, X, y)
    201         """
    202         K = self._get_kernel(X)
--> 203         self._fit_transform(K)
    204 
    205         if self.fit_inverse_transform:

C:\Users\gamer\Anaconda3\lib\site-packages\sklearn\decomposition\kernel_pca.py in _fit_transform(self, K)
    138         """ Fit's using kernel K"""
    139         # center kernel
--> 140         K = self._centerer.fit_transform(K)
    141 
    142         if self.n_components is None:

C:\Users\gamer\Anaconda3\lib\site-packages\sklearn\base.py in fit_transform(self, X, y, **fit_params)
    453         if y is None:
    454             # fit method of arity 1 (unsupervised transformation)
--> 455             return self.fit(X, **fit_params).transform(X)
    456         else:
    457             # fit method of arity 2 (supervised transformation)

C:\Users\gamer\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py in transform(self, K, y, copy)
   1530         check_is_fitted(self, 'K_fit_all_')
   1531 
-> 1532         K = check_array(K, copy=copy, dtype=FLOAT_DTYPES)
   1533 
   1534         K_pred_cols = (np.sum(K, axis=1) /

C:\Users\gamer\Anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, accept_sparse, dtype, order, copy, force_all_finite, ensur
e_2d, allow_nd, ensure_min_samples, ensure_min_features, warn_on_dtype, estimator)
    371                                        force_all_finite)
```

```
    372     else:
--> 373         array = np.array(array, dtype=dtype, order=order, copy=copy)
    374 
    375         if ensure_2d:
```

```
MemoryError: 
```

Проблему с MemoryError победить так и не удалось, на машине с 16гб оперативной памяти и 1.5тб выделенного свопа, испробованные различные ухищрения результата не дали =(
Поэтому дальше будем оперировать 10% данных для удобства и скорости

In [80]:

```
clf = KernelPCA(n_components=10, kernel='poly', degree=2)
clf.fit(x[:4200])
```

Out[80]:

```
KernelPCA(alpha=1.0, coef0=1, degree=2, eigen_solver='auto',
     fit_inverse_transform=False, gamma=None, kernel='poly',
     kernel_params=None, max_iter=None, n_components=10,
     remove_zero_eig=False, tol=0)
```

In [81]:

```
kernelPCA_degree2_data = clf.transform(x[:4200])
```

In [83]:

```
X_train, X_test, y_train, y_test = sklearn.cross_validation.train_test_split(kernelPCA_degree2_data, y[:4200],
                                                                              test_size=0.2, random_state=0)
```

In [84]:

```
n_jobs = -1

print("Started to fitting KNN on data with %d cores..." % 8 if n_jobs == -1 else n_jobs)

accuracy_values = []
for number in range(1, 10):
    t0 = time()

    clf = neighbors.KNeighborsClassifier(number, n_jobs=n_jobs, weights='uniform')

    clf.fit(X_train, y_train)

    y_predict = clf.predict(X_test)

    temp_acc = accuracy_score(y_test, y_predict)
    print('We have ', temp_acc, 'of accuracy on KNN with ', number, ' neighbor')
    accuracy_values.append(temp_acc)
    print("done in %0.3fs" % (time() - t0))
    print()
```

```
Started to fitting KNN on data with 8 cores...
We have  0.853571428571 of accuracy on KNN with  1  neighbor
done in 0.662s

We have  0.842857142857 of accuracy on KNN with  2  neighbor
done in 0.146s

We have  0.855952380952 of accuracy on KNN with  3  neighbor
done in 0.168s

We have  0.863095238095 of accuracy on KNN with  4  neighbor
done in 0.127s

We have  0.860714285714 of accuracy on KNN with  5  neighbor
done in 0.123s

We have  0.860714285714 of accuracy on KNN with  6  neighbor
done in 0.120s

We have  0.859523809524 of accuracy on KNN with  7  neighbor
done in 0.121s

We have  0.854761904762 of accuracy on KNN with  8  neighbor
done in 0.126s

We have  0.857142857143 of accuracy on KNN with  9  neighbor
done in 0.135s
```

In [85]:

```
n_jobs = -1

print("Started to fitting ExtraTreesClassifier on data with %d cores..." % 8 if n_jobs == -1 else n_jobs)

t0 = time()

forest = ExtraTreesClassifier(n_estimators=2000,
                              max_features=9,
                              n_jobs=n_jobs,
                              random_state=0)

forest.fit(X_train, y_train)

y_predict = forest.predict(X_test)

print('we have ', accuracy_score(y_test, y_predict), 'of accuracy on ExtraTreeClassifier')

print("done in %0.3fs" % (time() - t0))
```

```
Started to fitting ExtraTreesClassifier on data with 8 cores...
we have  0.867857142857 of accuracy on ExtraTreeClassifier
done in 26.614s
```

Имеем ухудшение качества отноительно классического линейного PCA

Попробуем различные степени

In [88]:

```
acc_value_knn = []
acc_value_extrtr = []

for deg in log_progress(range(3, 10)):

    clf = KernelPCA(n_components=10, kernel='poly', degree=deg)
    clf.fit(x[:4200])
    X_train, X_test, y_train, y_test = sklearn.cross_validation.train_test_split(clf.transform(x[:4200]), y[:4200],
                                                                                 test_size=0.2, random_state=0)

    n_jobs = -1

    print("Started to fitting KNN on data with %d cores..." % 8 if n_jobs == -1 else n_jobs)

    accuracy_values = []

    for number in range(1, 10):
        t0 = time()

        clf = neighbors.KNeighborsClassifier(number, n_jobs=n_jobs, weights='uniform')

        clf.fit(X_train, y_train)

        y_predict = clf.predict(X_test)

        temp_acc = accuracy_score(y_test, y_predict)
        print('We have ', temp_acc, 'of accuracy on KNN with ', number, ' neighbor')
        accuracy_values.append(temp_acc)
        print("done in %0.3fs" % (time() - t0))
        print()

    acc_value_knn.append(accuracy_values)

    n_jobs = -1

    print("Started to fitting ExtraTreesClassifier on data with %d cores..." % 8 if n_jobs == -1 else n_jobs)

    t0 = time()

    forest = ExtraTreesClassifier(n_estimators=2000,
                                  max_features=9,
                                  n_jobs=n_jobs,
                                  random_state=0)

    forest.fit(X_train, y_train)

    y_predict = forest.predict(X_test)

    score = accuracy_score(y_test, y_predict)

    acc_value_extrtr.append(score)

    print('we have ', score, 'of accuracy on ExtraTreeClassifier')

    print("done in %0.3fs" % (time() - t0))
```

```
Started to fitting KNN on data with 8 cores...
We have  0.836904761905 of accuracy on KNN with  1  neighbor
done in 0.123s

We have  0.832142857143 of accuracy on KNN with  2  neighbor
done in 0.117s

We have  0.845238095238 of accuracy on KNN with  3  neighbor
done in 0.137s

We have  0.84880952381 of accuracy on KNN with  4  neighbor
done in 0.117s

We have  0.839285714286 of accuracy on KNN with  5  neighbor
done in 0.132s

We have  0.853571428571 of accuracy on KNN with  6  neighbor
done in 0.121s

We have  0.853571428571 of accuracy on KNN with  7  neighbor
done in 0.130s

We have  0.847619047619 of accuracy on KNN with  8  neighbor
done in 0.120s

We have  0.840476190476 of accuracy on KNN with  9  neighbor
done in 0.149s

Started to fitting ExtraTreesClassifier on data with 8 cores...
we have  0.858333333333 of accuracy on ExtraTreeClassifier
done in 5.263s
Started to fitting KNN on data with 8 cores...
We have  0.802380952381 of accuracy on KNN with  1  neighbor
done in 0.146s

We have  0.807142857143 of accuracy on KNN with  2  neighbor
done in 0.115s

We have  0.82380952381 of accuracy on KNN with  3  neighbor
done in 0.131s

We have  0.830952380952 of accuracy on KNN with  4  neighbor
done in 0.118s

We have  0.830952380952 of accuracy on KNN with  5  neighbor
done in 0.130s

We have  0.828571428571 of accuracy on KNN with  6  neighbor
done in 0.121s
```

```
We have  0.82619047619 of accuracy on KNN with  7  neighbor
done in 0.132s

We have  0.82619047619 of accuracy on KNN with  8  neighbor
done in 0.120s

We have  0.82380952381 of accuracy on KNN with  9  neighbor
done in 0.127s

Started to fitting ExtraTreesClassifier on data with 8 cores...
we have  0.847619047619 of accuracy on ExtraTreeClassifier
done in 5.237s
Started to fitting KNN on data with 8 cores...
We have  0.791666666667 of accuracy on KNN with  1  neighbor
done in 0.148s

We have  0.778571428571 of accuracy on KNN with  2  neighbor
done in 0.117s

We have  0.808333333333 of accuracy on KNN with  3  neighbor
done in 0.118s

We have  0.810714285714 of accuracy on KNN with  4  neighbor
done in 0.127s

We have  0.810714285714 of accuracy on KNN with  5  neighbor
done in 0.123s

We have  0.803571428571 of accuracy on KNN with  6  neighbor
done in 0.121s

We have  0.813095238095 of accuracy on KNN with  7  neighbor
done in 0.126s

We have  0.803571428571 of accuracy on KNN with  8  neighbor
done in 0.127s

We have  0.80119047619 of accuracy on KNN with  9  neighbor
done in 0.122s

Started to fitting ExtraTreesClassifier on data with 8 cores...
we have  0.825 of accuracy on ExtraTreeClassifier
done in 7.002s
Started to fitting KNN on data with 8 cores...
We have  0.771428571429 of accuracy on KNN with  1  neighbor
done in 0.130s

We have  0.754761904762 of accuracy on KNN with  2  neighbor
done in 0.131s

We have  0.752380952381 of accuracy on KNN with  3  neighbor
done in 0.121s

We have  0.772619047619 of accuracy on KNN with  4  neighbor
done in 0.121s

We have  0.77380952381 of accuracy on KNN with  5  neighbor
done in 0.118s

We have  0.775 of accuracy on KNN with  6  neighbor
done in 0.123s

We have  0.77380952381 of accuracy on KNN with  7  neighbor
done in 0.123s

We have  0.772619047619 of accuracy on KNN with  8  neighbor
done in 0.124s

We have  0.763095238095 of accuracy on KNN with  9  neighbor
done in 0.122s

Started to fitting ExtraTreesClassifier on data with 8 cores...
we have  0.795238095238 of accuracy on ExtraTreeClassifier
done in 6.446s
Started to fitting KNN on data with 8 cores...
We have  0.716666666667 of accuracy on KNN with  1  neighbor
done in 0.115s

We have  0.709523809524 of accuracy on KNN with  2  neighbor
done in 0.118s

We have  0.717857142857 of accuracy on KNN with  3  neighbor
done in 0.121s

We have  0.729761904762 of accuracy on KNN with  4  neighbor
done in 0.119s

We have  0.736904761905 of accuracy on KNN with  5  neighbor
done in 0.121s

We have  0.727380952381 of accuracy on KNN with  6  neighbor
done in 0.124s

We have  0.725 of accuracy on KNN with  7  neighbor
done in 0.121s

We have  0.714285714286 of accuracy on KNN with  8  neighbor
done in 0.131s

We have  0.722619047619 of accuracy on KNN with  9  neighbor
done in 0.126s

Started to fitting ExtraTreesClassifier on data with 8 cores...
we have  0.764285714286 of accuracy on ExtraTreeClassifier
done in 5.513s
Started to fitting KNN on data with 8 cores...
We have  0.655952380952 of accuracy on KNN with  1  neighbor
done in 0.130s

We have  0.664285714286 of accuracy on KNN with  2  neighbor
```

```
done in 0.116s

We have  0.664285714286 of accuracy on KNN with  3  neighbor
done in 0.124s

We have  0.666666666667 of accuracy on KNN with  4  neighbor
done in 0.123s

We have  0.663095238095 of accuracy on KNN with  5  neighbor
done in 0.123s

We have  0.671428571429 of accuracy on KNN with  6  neighbor
done in 0.123s

We have  0.670238095238 of accuracy on KNN with  7  neighbor
done in 0.125s

We have  0.67619047619 of accuracy on KNN with  8  neighbor
done in 0.127s

We have  0.664285714286 of accuracy on KNN with  9  neighbor
done in 0.125s

Started to fitting ExtraTreesClassifier on data with 8 cores...
we have  0.733333333333 of accuracy on ExtraTreeClassifier
done in 5.682s
Started to fitting KNN on data with 8 cores...
We have  0.603571428571 of accuracy on KNN with  1  neighbor
done in 0.119s

We have  0.610714285714 of accuracy on KNN with  2  neighbor
done in 0.115s

We have  0.603571428571 of accuracy on KNN with  3  neighbor
done in 0.127s

We have  0.607142857143 of accuracy on KNN with  4  neighbor
done in 0.116s

We have  0.608333333333 of accuracy on KNN with  5  neighbor
done in 0.136s

We have  0.604761904762 of accuracy on KNN with  6  neighbor
done in 0.834s

We have  0.6 of accuracy on KNN with  7  neighbor
done in 0.132s

We have  0.602380952381 of accuracy on KNN with  8  neighbor
done in 0.121s

We have  0.588095238095 of accuracy on KNN with  9  neighbor
done in 0.134s

Started to fitting ExtraTreesClassifier on data with 8 cores...
we have  0.715476190476 of accuracy on ExtraTreeClassifier
done in 6.258s
```

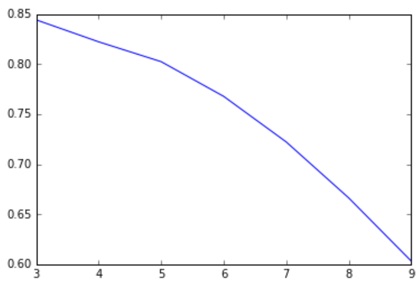
Посмотрим на качество при различных степенях ядра для KNN

In [91]:

```
plt.plot([i for i in range(3,10)],[np.mean(acc_value_knn[i]) for i in range(7)])
```

Out[91]:

```
[<matplotlib.lines.Line2D at 0x13d01373da0>]
```



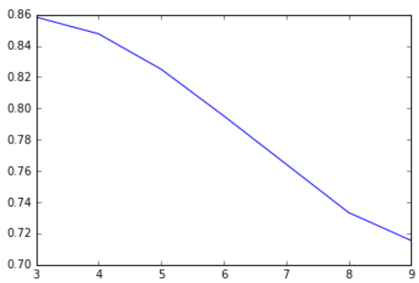
Как видно, с ростом степени ядра имеем только ухудшение для метода ближайших соседей

In [93]:

```
plt.plot([i for i in range(3,10)], acc_value_extrtr)
```

Out[93]:

```
[<matplotlib.lines.Line2D at 0x13d00bcae10>]
```



Аналогичные показатели для леса решающих деревьев

Попробуем Гауссову радиальную базисную функцию

In [94]:

```
clf = KernelPCA(n_components=10, kernel='rbf')
clf.fit(x[:4200])
X_train, X_test, y_train, y_test = sklearn.cross_validation.train_test_split(clf.transform(x[:4200]), y[:4200],
                                                                             test_size=0.2, random_state=0)

n_jobs = -1

print("Started to fitting KNN on data with %d cores..." % 8 if n_jobs == -1 else n_jobs)

accuracy_values = []

for number in range(1, 10):
    t0 = time()

    clf = neighbors.KNeighborsClassifier(number, n_jobs=n_jobs, weights='uniform')

    clf.fit(X_train, y_train)

    y_predict = clf.predict(X_test)

    temp_acc = accuracy_score(y_test, y_predict)
    print('We have ', temp_acc, 'of accuracy on KNN with ', number, ' neighbor')
    accuracy_values.append(temp_acc)
    print("done in %0.3fs" % (time() - t0))
    print()

n_jobs = -1

print("Started to fitting ExtraTreesClassifier on data with %d cores..." % 8 if n_jobs == -1 else n_jobs)

t0 = time()

forest = ExtraTreesClassifier(n_estimators=2000,
                              max_features=9,
                              n_jobs=n_jobs,
                              random_state=0)

forest.fit(X_train, y_train)

y_predict = forest.predict(X_test)

score = accuracy_score(y_test, y_predict)

print('we have ', score, 'of accuracy on ExtraTreeClassifier')

print("done in %0.3fs" % (time() - t0))
```

```
Started to fitting KNN on data with 8 cores...
We have  0.107142857143 of accuracy on KNN with  1  neighbor
done in 0.119s

We have  0.102380952381 of accuracy on KNN with  2  neighbor
done in 0.114s

We have  0.0964285714286 of accuracy on KNN with  3  neighbor
done in 0.129s

We have  0.0988095238095 of accuracy on KNN with  4  neighbor
done in 0.123s

We have  0.107142857143 of accuracy on KNN with  5  neighbor
done in 0.129s

We have  0.104761904762 of accuracy on KNN with  6  neighbor
done in 0.121s

We have  0.109523809524 of accuracy on KNN with  7  neighbor
done in 0.134s

We have  0.113095238095 of accuracy on KNN with  8  neighbor
done in 0.119s

We have  0.117857142857 of accuracy on KNN with  9  neighbor
done in 0.126s

Started to fitting ExtraTreesClassifier on data with 8 cores...
we have  0.0964285714286 of accuracy on ExtraTreeClassifier
done in 5.849s
```

Имеем просто ужасное качество, но хорошую скорость =(

Напоследок, результат полученный в ходе модификации сверточной нейронной сети

In [ ]:

```
IMAGE_SIZE = 28
NUM_CHANNELS = 1
NUM_LABELS = 10
VALIDATION_SIZE = 1000
SEED = 66478
BATCH_SIZE = 64
NUM_EPOCHS = 10
EVAL_BATCH_SIZE = 64
EVAL_FREQUENCY = 100

def error_rate(predictions, labels):
  return 100.0 - (
      100.0 *
      numpy.sum(numpy.argmax(predictions, 1) == labels) /
      predictions.shape[0])

def main(argv=None):
```

```python
def main(argv=None):
  train = pandas.read_csv('train.csv')
  train_data = []
  for i in range(42000):
    temp = list(convert(train[i:i + 1])[0])
    temp = np.array(temp)

    temp.shape = ((28), (28), (1))
    temp = (temp - (255 / 2.0)) / 255
    #train_data.append(list(convert(train_set[i:i + 1])[0]))
    train_data.append(temp)
  train_labels = train.label

  test = pandas.read_csv('test.csv')
  test_data = []
  for i in range(28000):
    temp = list(test[i:i + 1].iloc[:,0:].values[0])
    temp = np.array(temp)

    temp.shape = ((28), (28), (1))
    temp = (temp - (255 / 2.0)) / 255
    test_data.append(temp)

  train_data = np.array(train_data)
  test_data = np.array(test_data)
  validation_data = train_data[:VALIDATION_SIZE, ...]
  validation_labels = train_labels[:VALIDATION_SIZE]
  train_data = train_data[VALIDATION_SIZE:, ...]
  train_labels = train_labels[VALIDATION_SIZE:]
  num_epochs = NUM_EPOCHS
  train_size = train_labels.shape[0]

  train_data_node = tf.placeholder(
      tf.float32,
      shape=(BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, NUM_CHANNELS))
  train_labels_node = tf.placeholder(tf.int64, shape=(BATCH_SIZE,))
  eval_data = tf.placeholder(
      tf.float32,
      shape=(EVAL_BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, NUM_CHANNELS))

  conv1_weights = tf.Variable(
      tf.truncated_normal([5, 5, NUM_CHANNELS, 32],
                          stddev=0.1,
                          seed=SEED))
  conv1_biases = tf.Variable(tf.zeros([32]))
  conv2_weights = tf.Variable(
      tf.truncated_normal([5, 5, 32, 64],
                          stddev=0.1,
                          seed=SEED))
  conv2_biases = tf.Variable(tf.constant(0.1, shape=[64]))
  fc1_weights = tf.Variable(
      tf.truncated_normal(
          [IMAGE_SIZE // 4 * IMAGE_SIZE // 4 * 64, 512],
          stddev=0.1,
          seed=SEED))
  fc1_biases = tf.Variable(tf.constant(0.1, shape=[512]))
  fc2_weights = tf.Variable(
      tf.truncated_normal([512, NUM_LABELS],
                          stddev=0.1,
                          seed=SEED))
  fc2_biases = tf.Variable(tf.constant(0.1, shape=[NUM_LABELS]))

  def model(data, train=False):
    conv = tf.nn.conv2d(data,
                        conv1_weights,
                        strides=[1, 1, 1, 1],
                        padding='SAME')
    relu = tf.nn.relu(tf.nn.bias_add(conv, conv1_biases))

    pool = tf.nn.max_pool(relu,
                          ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1],
                          padding='SAME')
    conv = tf.nn.conv2d(pool,
                        conv2_weights,
                        strides=[1, 1, 1, 1],
                        padding='SAME')
    relu = tf.nn.relu(tf.nn.bias_add(conv, conv2_biases))
    pool = tf.nn.max_pool(relu,
                          ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1],
                          padding='SAME')
    pool_shape = pool.get_shape().as_list()
    reshape = tf.reshape(
        pool,
        [pool_shape[0], pool_shape[1] * pool_shape[2] * pool_shape[3]])
    hidden = tf.nn.relu(tf.matmul(reshape, fc1_weights) + fc1_biases)

    if train:
      hidden = tf.nn.dropout(hidden, 0.5, seed=SEED)
    return tf.matmul(hidden, fc2_weights) + fc2_biases

  logits = model(train_data_node, True)
  loss = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
      logits, train_labels_node))

  regularizers = (tf.nn.l2_loss(fc1_weights) + tf.nn.l2_loss(fc1_biases) +
                  tf.nn.l2_loss(fc2_weights) + tf.nn.l2_loss(fc2_biases))
  loss += 5e-4 * regularizers

  batch = tf.Variable(0)
  learning_rate = tf.train.exponential_decay(
      0.01,
      batch * BATCH_SIZE,
      train_size,
      0.95,
      staircase=True)

  optimizer = tf.train.MomentumOptimizer(learning_rate,
                                         0.9).minimize(loss,
```

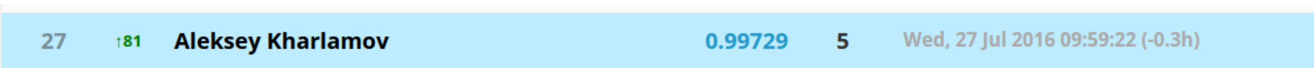
```
                                                    global_step=batch)

  train_prediction = tf.nn.softmax(logits)

  eval_prediction = tf.nn.softmax(model(eval_data))

  def eval_in_batches(data, sess):
    size = data.shape[0]
    if size < EVAL_BATCH_SIZE:
      raise ValueError("batch size for evals larger than dataset: %d" % size)
    predictions = numpy.ndarray(shape=(size, NUM_LABELS), dtype=numpy.float32)
    for begin in xrange(0, size, EVAL_BATCH_SIZE):
      end = begin + EVAL_BATCH_SIZE
      if end <= size:
        predictions[begin:end, :] = sess.run(
            eval_prediction,
            feed_dict={eval_data: data[begin:end, ...]})
      else:
        batch_predictions = sess.run(
            eval_prediction,
            feed_dict={eval_data: data[-EVAL_BATCH_SIZE:, ...]})
        predictions[begin:, :] = batch_predictions[begin - size:, :]
    return predictions

  start_time = time.time()
  with tf.Session() as sess:
    tf.initialize_all_variables().run()
    print('Initialized!')
    for step in xrange(int(num_epochs * train_size) // BATCH_SIZE):
      offset = (step * BATCH_SIZE) % (train_size - BATCH_SIZE)
      batch_data = train_data[offset:(offset + BATCH_SIZE), ...]
      batch_labels = train_labels[offset:(offset + BATCH_SIZE)]
      feed_dict = {train_data_node: batch_data,
                   train_labels_node: batch_labels}

      _, l, lr, predictions = sess.run(
          [optimizer, loss, learning_rate, train_prediction],
          feed_dict=feed_dict)
      if step % EVAL_FREQUENCY == 0:
        elapsed_time = time.time() - start_time
        start_time = time.time()
        print('Step %d (epoch %.2f), %.1f ms' %
              (step, float(step) * BATCH_SIZE / train_size,
               1000 * elapsed_time / EVAL_FREQUENCY))
        print('Minibatch loss: %.3f, learning rate: %.6f' % (l, lr))
        print('Minibatch error: %.1f%%' % error_rate(predictions, batch_labels))
        print('Validation error: %.1f%%' % error_rate(
            eval_in_batches(validation_data, sess), validation_labels))
        sys.stdout.flush()

    print('Final!')
    out = eval_in_batches(test_data, sess)
    pandas.DataFrame(out).to_csv('Kaggle.csv')

if __name__ == '__main__':
  tf.app.run()
```

По результатам на Kaggle имеем 27 место из 1114 участников (top 3%) с accuracy 0.99729.

In [5]:

```
Image(url= "Screen1.png")
```

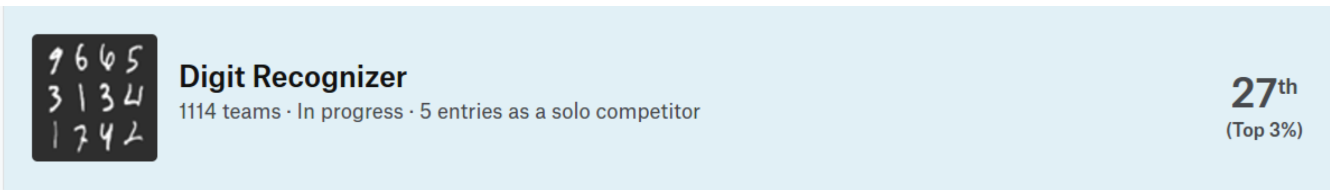
Out[5]:



In [6]:

```
Image(url="Screen2.png")
```

Out[6]:



Заключение

В ходе выполнения летней практики было приобретено множество знаний связанных с Machine Learning, Feature Engineering, Principal Component Analysis, что несомненно является хорошим подспорьем для дальнейшего изучения науки о данных. Во время исследования различных классификаторов был получен опыт в применении таких методов, как:

1. KNN (метод k - ближайших соседей) с различными метриками.
2. Support Vector Machine, в частности с применением различных ядерных хаков.
3. Деревья решений.
4. Random Forest.
5. Gradient boosting of decision tree.
6. AdaBoost.
7. Генетические алгоритмы подбора оптимальных параметров, такие как TPOT.
8. Многослойный перцептрон.
9. Convolution neural network.

Полученные навыки являются необходимыми для желанной мною для дальнейшего изучения специализации Machine Learning, поэтому они являются чрезвычайно полезными.

В дальнейшем хотелось бы применить полученные знания в других соревнованиях, проводимых на площадке Kaggle, так как в исследовании поставленной выше задачи получилось достичь некоторых результатов.