Отчет по практическому заданию: метрические методы классификации

Выполнено студентом 317 группы, Филимоновым Владиславом

1 Введение

В данном отчете будет проведен анализ экспериментов с dataset'ом MNIST. Подробное описание условий экспериментов можно найти в [1].

2 Эксперименты

2.1 Сравнение времени работы KNN алгоритма в зависимости от стратегии поиска ближайших соседей

Были рассмотрены четыре стратегии поиска: brute, kd-tree, ball-tree и my own, реализованная в модуле KNNClassifier.py. Из графика(Рис.1) видно,что самым эффективным по времени является стратегия brute. В дальнейших экспериментах будет использована стратегия поиска brute, так как выбор стратегии(среди точных стратегий) влияет только на быстродействие алгоритма и не влияет на точность.

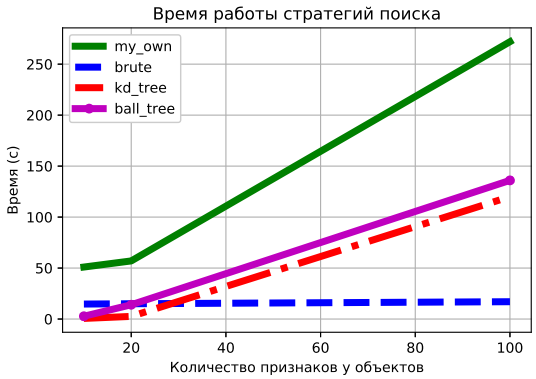

Рис. 1: Время работы стратегий поиска

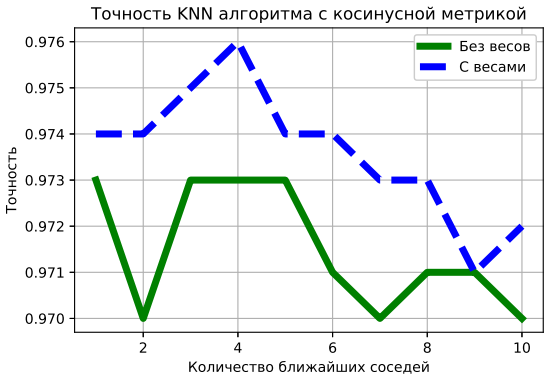

Рис. 2: Точность KNN с косинусной метрикой

2.2 Оценка точности алгоритма KNN без весов в зависимости от гиперпараметра K и метрики

В этом эксперименте была проведена кросс-валидация на трех фолдах, для алгоритмов KNN без весов с K $\in \{1, \ldots, 10\}$ и евклидовой или косинусной метрикой. Для каждого алгоритма была посчитана точность на каждом фолде, но так как информация о точности на определенном фолде не является достаточно информативной, были рассмотрены максимальные и средние точности по трем фолдам. Все средние и максимальные точности алгоритма с различными K и метриками представлены ниже(жирным шрифтом выделены наибольшие значения). Все значения были округлены до третьего знака после запятой, точные значения можно найти в приложенном файле experiement.ipynb.

Из таблицы 1 следует, что максимальная и средняя точность отличаются незначительно и достигают максимума одновременно, и что наибольшая точность достигается при $k \in \{3, 4\}$ и косинусной метрике.

2.3 Оценка точности алгоритма KNN с весами в зависимости от гиперпараметра K и метрики

Этот эксперимент проводился аналогично эксперименту, описанному в предыдущем разделе, за исключением того, что в этом эксперименте KNN алгоритм использовал веса. Ниже приводится

Таблица 1: Точность алгоритма KNN на кросс-валидации

	Евклидова метрика		Косинусная метрика	
k	Максимальная	Средняя	Максимальная	Средняя
1	**0.971**	**0.970**	0.974	0.973
2	0.965	0.964	0.971	0.970
3	**0.971**	**0.970**	**0.975**	**0.973**
4	0.970	0.969	**0.975**	**0.973**
5	0.970	0.969	0.973	0.973
6	0.969	0.967	0.974	0.971
7	0.969	0.967	0.973	0.970
8	0.966	0.966	0.972	0.971
9	0.966	0.965	0.971	0.971
10	0.966	0.968	0.971	0.970

аналогичная таблица для данного алгоритма.

Из таблицы 2 следует, что максимальная и средняя точность отличаются незначительно и достигают максимума одновременно, и что наибольшая точность достигается при $k = 4$ и косинусной метрике. Из сравнения таблиц 1 и 2 можно сделать вывод, что алгоритм с весами имеет меньшее отклонение значения максимальной точности от средней.

Таблица 2: Точность KNN с весами на кросс-валидации

	Евклидова метрика		Косинусная метрика	
k	Максимальная	Средняя	Максимальная	Средняя
1	0.970	0.970	0.974	0.974
2	0.970	0.970	0.974	0.974
3	0.971	0.971	0.975	0.975
4	**0.972**	**0.972**	**0.976**	**0.976**
5	0.970	0.970	0.974	0.974
6	0.971	0.970	0.975	0.974
7	0.969	0.969	0.973	0.973
8	0.970	0.968	0.973	0.973
9	0.968	0.967	0.972	0.971
10	0.966	0.963	0.972	0.972

Для определения оптимальных значений гиперпараметра K и иллюстрации сравнения метода без весов и с весами построен график(Рис.2) средней точности от количества ближайший соседей k. Проанализоровав график, приходим к выводу, что наибольшая точность на кросс-валидации достигается у KNN алгоритма с весами, $k = 4$ и косинусной метрикой.

2.4 Анализ работы лучшего алгоритма на тестовых данных

В этом эксперименте оценивается точность работы алгоритма с оптимальными гиперпараметрами, подобранными по кросс-валидации в разделах 2.2 и 2.3. Точность работы алгоритма на тестовых данных $accuracy_{test} = 0.975$ и средняя точность на кросс валидации $accuracy_{cv} = 0.976$ достаточно близки.

Точность данного алгоритма значительно ниже, чем точность лучших алгоритмов, полный список которых можно найти в [2]. Например, алгоритм Regularization of Neural Networks using DropConnect имеет точность 0.9979, а лучший алгоритм, использующий идею KNN - Large-Margin kNN Classification using a Deep Encoder Network имеет точность 0.9906.

Для анализа ошибок алгоритма рассмотрим confusion matrix(Рис. 3). Для ее визуализации использовался код, приложенный в файле plot_confusion_matrix.py, взятый из [3].

Рассмотрев Рис.3, можно сделать вывод, для каждого класса существуют несколько цифр на которых достигается наибольшая ошибка предсказания. Например, для элементов, которые алгоритм относит к классу 0, наиболее частые ошибочные предсказания это $\{2, 8, 9\}$. Несложно заметить, что 0, 2, 8, 9 при рукописном написании имеют наиболее ярко выраженную общую особенность - округлость, малое количество острых углов. Но при этом, для цифр, которые алгоритм относит к 2, наибольшая ошибка происходит на элементах 3 и 7, а на элементах 0 ошибки нет. Это можно объяснить тем, что сходство 2 и 7 более ярко выражено, чем сходство 2 и 0. Таким образом, для каждого класса основная доля ошибок предсказания приходится на цифры, имеющие наиболее похожее с меткой класса рукописное написание.

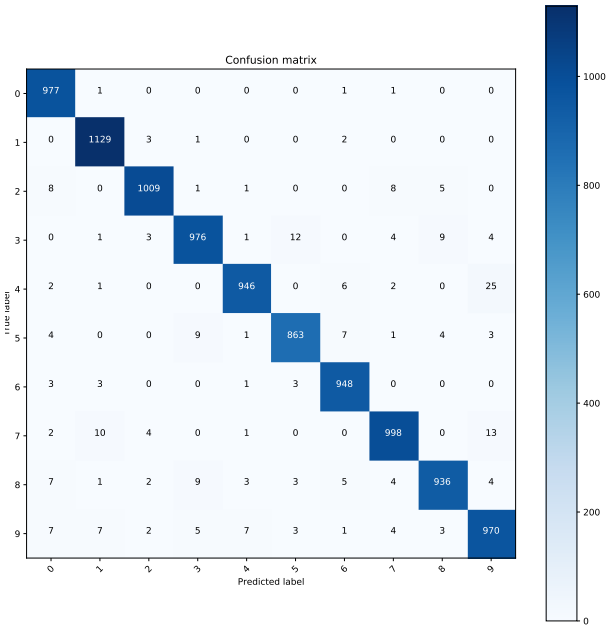

Рис. 3: Confusion matrix

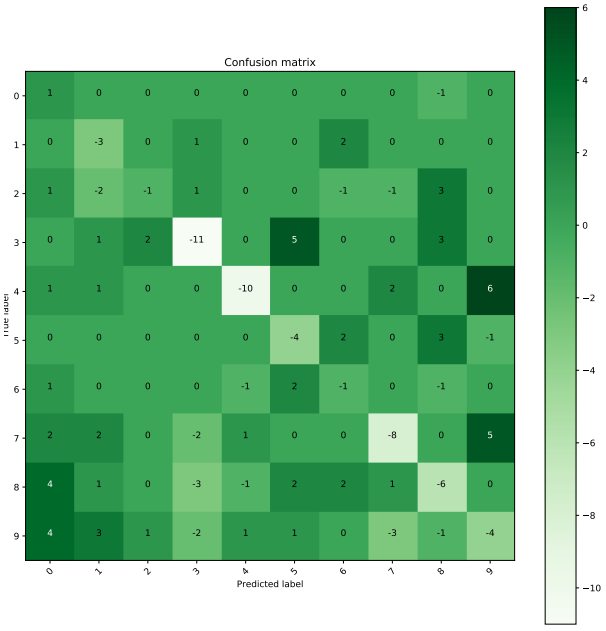

Рис. 4: Разница Confusion matrix после поворота

Рассмотрим несколько объектов, на которых была допущена ошибка. На рис. 5 - 8 изображены объекты, на которых была допущена ошибка, при этом в подписи к рисунку цифра, заключенная в (), это метка предсказанного класса. Можно заметить, что их объединяет нечеткое написание, например, отсутсвие горизонтальной палочки у 7, длинного хвоста снизу у 2, большой окружности(или эллипса) у 8 снизу и слишком мелкое написание 9.

2.5 Аугментация обучающей выборки

В этом эксперименте была размножена обучающая выборка с помошью поворотов и сдвигов изображений, а так же применения фильтра Гаусса к изображениям.

Рис. 5: Цифра 7(1)

Рис. 6: Цифра 9(4)

Рис. 7: Цифра 2(7)

Рис. 8: Цифра 8(9)

По кросс-валидации были найдены оптимальные параметры для этих преобразований. Для реализации кросс-валидации была написана функция knn_cross_val_score_for5, которая описана в модуле cross_validation.py. Эта функция реализована таким образом, что в любой момент времени она не хранит не более чем 2 обучающие выборки и не более чем 2 матрицы ближайших соседей, что является оптимальным решением по памяти.

В таблице 3 приведены максимальные и средние точности на кросс-валидации по трем фолдам для каждого преобразования, при этом параметр для поворота это градусы, для сдвига - пиксели, для Гауссовского фильтра - дисперсия. Поворот производился по и против часовой стрелки, а сдвиг - вверх, вниз, вправо и влево. Все значения в таблице были округлены до третьего знака после запятой, неокругленные значения можно найти в приложенном файле experiement.ipynb. Жирным шрифтов выделены максимальные значения точности для каждого преобразования.

Для каждого преобразования, кроме фильтра, нашелся только один параметр, который максимизирует и среднюю, и максимальную точность, он и будет оптимальным. Для фильтра Гаусса в качестве оптимального параметра возьмем 1, так как неокругленное значение точности при этом параметре выше, чем неокругленное значение при 1.5.

Таблица 3: Точность KNN после аугментации обучающей выборки

Преобразование	Поворот			Сдвиг			Фильтр		
Параметр	5	10	15	1	2	3	0.5	1	1.5
Макс. точность	**0.979**	0.978	0.975	**0.975**	0.974	**0.975**	**0.979**	**0.979**	**0.979**
Средняя точность	**0.978**	**0.978**	0.975	**0.975**	0.974	0.974	0.978	**0.979**	**0.979**

Для анализа того, какие ошибки исправляет каждое преобразование, для каждого преобразования была построена Confusion matrix. Затем для удобства анализа была рассмотрена разница Confusion matrix для исходной выборки и Confusion matrix для выборки после каждого преобразования. При этом если элемент на главной диаголи отрицателен, то аугментация улучшила распознавание цифры, равной номеру строки данного элемента, если отрицателен - ухудшила распознавание данной цифры. Любой положительный элемент вне главной диагонали, говорит о том, что количество ошибок при распознавании цифры, равной номеру строки, как цифры, равной номеру столбца, уменьшилось.На рис. 4 приведена разница Confusion matrix для исходной выборки и Confusion matrix после поворота на угол в 5 градусов. Обозначим эту матрицу M^1. Так как $M^1_{ii} < 0$ для $\forall i \in \{1, \ldots, 9\}$ это преобразование улучшиа распознавание всех цифр, кроме 0. При этом значительно снизились ошибки распознавания 3 как 5, 4 как 9, и 7 как 9.

На рисунках 9 и 10 изображены матрицы, равные разнице исходной Confusion Matrix и Confusion Matrix после сдвига и применения фильтра соответсвенно. Обозначим матрицы изображенные на рисунках 9 и 10 M^2 и M^3 соответсвенно.

Так как $M^2_{ii} < 0$ для $\forall i \in \{1, \ldots, 9\}$ сдвиг улучшил распознавание всех цифр, кроме 0. При этом значительно снизились ошибки распознавания 3 как 5, 3 как 8, 4 как 6, 7 как 9 и 8 как 0.

Так как $M^3_{ii} < 0$ для $\forall i \in \{1, \ldots, 9\}$ применение фильтра улучшило распознавание всех цифр, кроме 0. При этом значительно снизились ошибки распознавания 3 как 5, 3 как 8, 4 как 9, 5 как 8, 8 как 0 и 9 как 0.

После применения всех преобразований с оптимальными параметрами точность выросла с 0.975 до 0.983. Для удобства анализа изменений Confusion Matrix после изменений, рассмотрим матрицу M, равную разнице Confusion Matrix до преобразований и Confusion Matrix после преобразований. Она изображена на рисунке 11.

Так как $M_{ii} < 0$ для $\forall i \in \{1, \ldots, 9\}$ аугментация обучающей выборки с помощью всех указанных преобразований с оптимальными параметрами улучшило распознавание всех цифр, кроме 0.

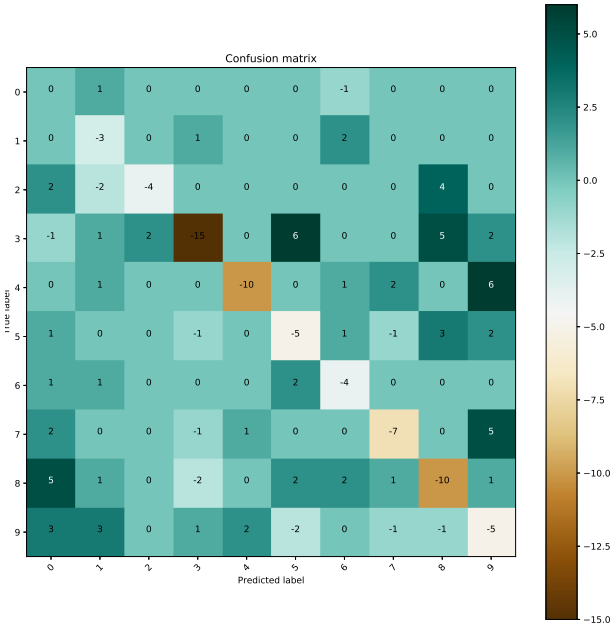

Рис. 9: Разница Confusion matrix после сдвига

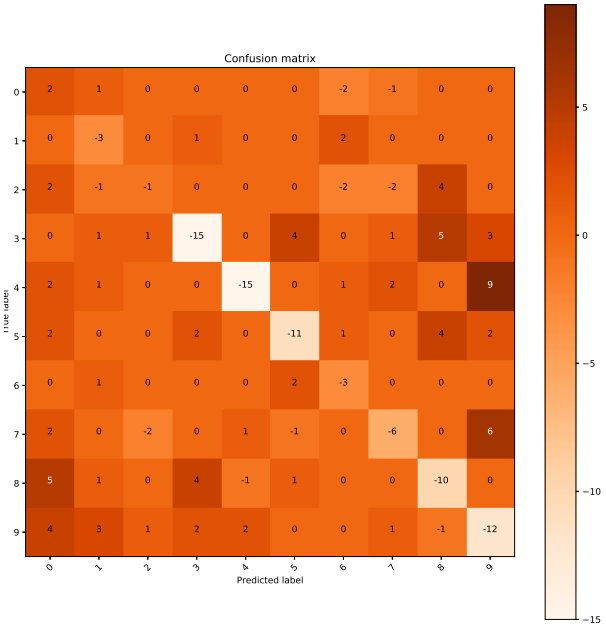

Рис. 10: Разница Confusion matrix после фильтра

При этом значительно снизились ошибки распознавания 3 как 5, 3 как 8, 4 как 9,5 как 8, 8 как 0, и 9 как 0.

2.6 Аугментация тестовой выборки

В этом эксперименте была размножена тестовая выборка с помошью аналогичных преобразований, описанных в предыдущем пункте. Вычисление ближайших соседей строилось по схеме, описанной в формулировке задания. При этом один и тот же объект обучающей выборки не может быть более чем одним из K ближайших соседей.

По кросс-валидации были найдены оптимальные параметры для этих преобразований. Для реализации кросс-валидации была написана функция knn_cross_val_score_for6, которая описана в модуле cross_validation.py. Эта функция реализована таким образом, что в любой момент времени она не хранит не более чем 2 тестовые выборки и не более чем 2 матрицы ближайших соседей, что является оптимальным решением по памяти.

В таблице 4 приведены максимальные и средние точности на кросс-валидации по трем фолдам для каждого преобразования, при этом параметр для поворота это градусы, для сдвига - пиксели, для Гауссовского фильтра - дисперсия. Поворот производился по и против часовой стрелки, а сдвиг - вверх, вниз, вправо и влево. Все значения в таблице были округлены до третьего знака после запятой, неокругленные значения можно найти в приложенном файле experiement.ipynb. Жирным шрифтов выделены максимальные значения точности для каждого преобразования.

Таблица 4: Точность KNN после аугментации тестовой выборки

Преобразование	Поворот			Сдвиг			Фильтр		
Параметр	5	10	15	1	2	3	0.5	1	1.5
Макс. точность	**0.976**	0.975	0.973	**0.979**	0.975	0.974	**0.970**	0.961	0.947
Средняя точность	**0.975**	0.974	0.971	**0.978**	0.975	0.974	**0.969**	0.960	0.946

Для каждого преобразования нашелся только один параметр, который максимизирует и среднюю, и максимальную точность, он и будет оптимальным. Но применение фильра Гаусса только

снижает точность, следовательно это преобразование не целесообразно применять.

Для анализа того, какие ошибки исправляет каждое преобразование используем метод описанный в разделе 2.5.

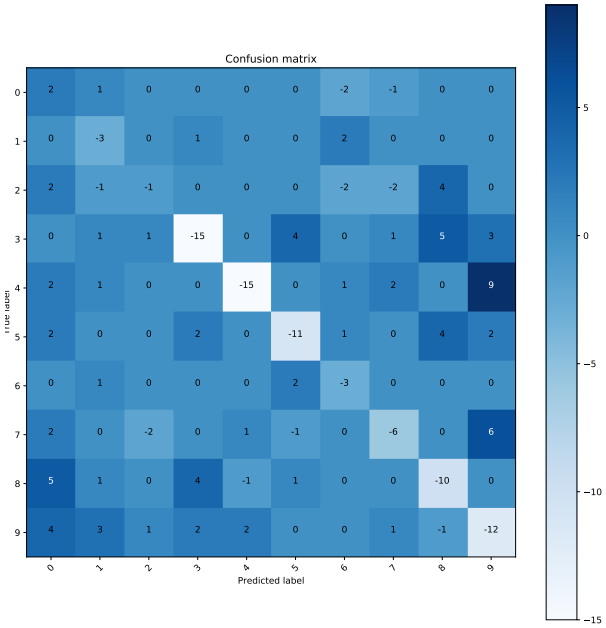

Рис. 11: Разница Confusion matrix после всех преобразований обучающей выборки

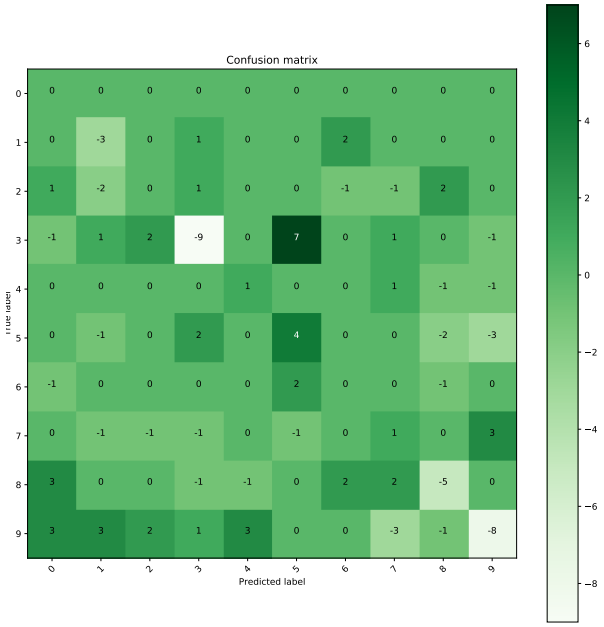

Рис. 12: Разница Confusion matrix после поворотов тестовой выборки

Обозначим матрицы изображенные на рисунках 12 и 13 M^1 и M^2 соответсвенно. Так как $M^1_{ii} < 0$ для $\forall i \in \{1, 3, 8, 9\}$ поворот улучшил распознавание 1, 3, 8, 9, при этом распознавание остальных цифр ухудшилось или не изменилось. При этом значительно снизились ошибки распознавания 3 как 5, 8 как 0, 9 как 0, 9 как 1 и 9 как 3.

Так как $M^2_{ii} < 0$ для $\forall i \in \{0, 1, 3, 4, 6, 7, 8, 9\}$ сдвиг улучшил распознавание всех цифр, кроме 2 и 5. При этом значительно снизились ошибки распознавания 3 как 5, 8 как 0, 9 как 1 и 9 как 4.

После применения всех преобразований с оптимальными параметрами точность выросла с 0.975 до 0.979. Для сравнения подходов, описанных в разделе 2.5 и в текущем разделе рассмотрим матрицу M, равную разнице Confusion Matrix после всех оптимальных преобразований, описанных в этом разделе и Confusion Matrix после всех оптимальных преобразований, описанных в разделе 2.5. Матрица M изображена на рисунке 14.

Так как $M_{ii} < 0$ для $\forall i \in \{0, \ldots, 9\}$ подход с аугментацией обучающей выборки сильнее улучшает распознавание всех цифр, чем подход описанный в текущем разделе. Что также подтверждается точностью на тестовой выборке: точность подхода, описанного в 2.5 на 0.004 выше, чем точность данного подхода.

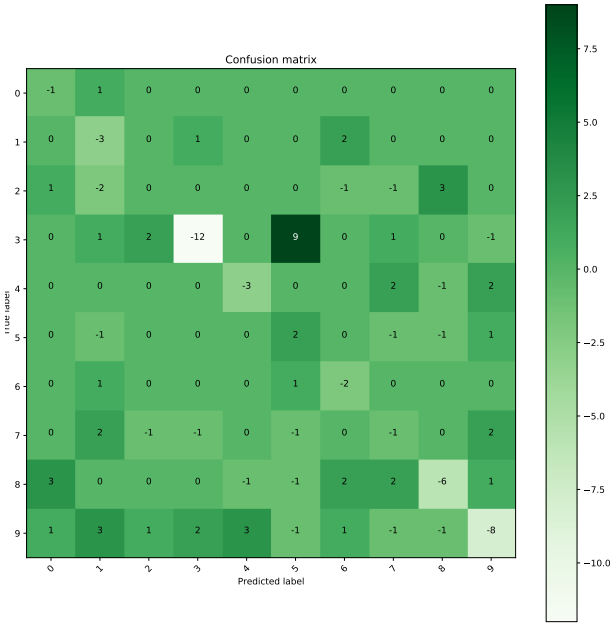

Рис. 13: Разница Confusion matrix после сдвигов тестовой выборки

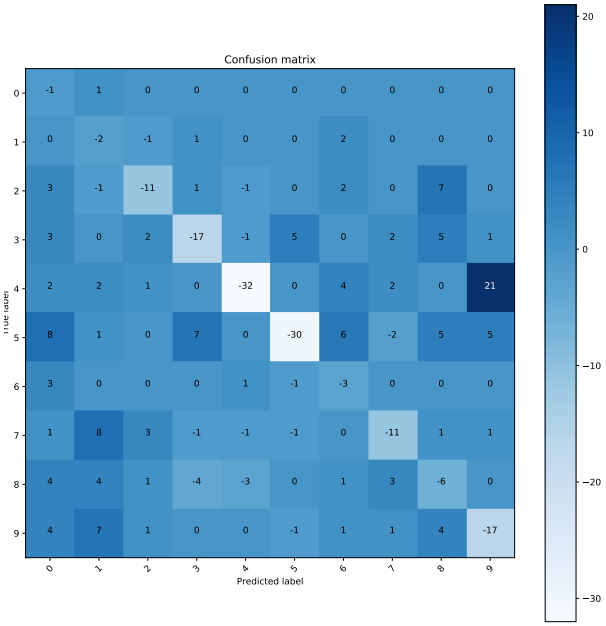

Рис. 14: Разница Confusion Matrix после всех преобразований обучающей и после всех преобразований тестовой выборок

Список литературы

[1] https://github.com/arti32lehtonen/mmp_prac_2017/blob/master/Tasks/task1.pdf

[2] http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.

[3] http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py