IST707 Applied Machine Learning

HW9: SVMs, kNN, and Random Forest for handwriting recognition

We will use Orange to implement SVMs, kNN, Random Forest, and Gradient Boosting algorithms for handwriting recognition, and compare their performance with the naïve Bayes and decision tree models we built in previous week.

Steps:

1. Data pre-processing steps:

    a. Using the Feature Statistic module, we just selected features that had a maximum value greater than zero. Features that have a maximum value of zero provide no information and got eliminated to reduce the noise.

    b. Using the Select Column module, we selected the label column as the target and all the other columns as features.

    c. Using the Edit Domain module, the type of label column is defined as categorical and all the other columns as numeric. Although the label column stores numeric values, here our intension is to classify these handwritings into nine categories.

    d. Impute module is used to eliminate rows with missing values. We can see that there are no rows with missing values in this data set because the input and output of this module have the same number of observations.

    e. After experimenting with all the ranking combinations, the best accuracy got achieved for all models while ranking the highest 602 features regarding the FCBF scoring method.

    f. After experimenting with all the discretization methods, we did not observe any improvement in the results and decided not to discretize the data because discretization increased the run time significantly.

2. Building and tuning kNN, SVM, Random Forest, and Gradient Boosting models:

    a. KNN: while tuning the model, cross-validation with 3-fold yielded the following information:

Number of neighbors	Metric	Weight	Precision	F1
17	Euclidean	Distance	.925	.921
17	Euclidean	Uniform	.922	.918
17	Manhattan	Distance	.912	.906

17	Manhattan	Uniform	.909	.901
17	Chebyshev	Distance	.689	.676
17	Chebyshev	Uniform	.670	.656
17	Mahalanobis	Distance	Error	Error
17	Mahalanobis	Uniform	Error	Error

Therefore, we decided to proceed with Euclidean Distance which showed the highest F1 value. Then, we changed the number of neighbors and got the following information:

Number of neighbors	Metric	Weight	Precision	F1
1	Euclidean	Distance	.936	.935
2	Euclidean	Distance	.936	.935
3	Euclidean	Distance	.939	.938
4	Euclidean	Distance	.941	.940
5	Euclidean	Distance	.941	.940
6	Euclidean	Distance	.940	.939
7	Euclidean	Distance	.938	.936
8	Euclidean	Distance	.938	.936
9	Euclidean	Distance	.935	.933
10	Euclidean	Distance	.935	.933
15	Euclidean	Distance	.928	.925
20	Euclidean	Distance	.922	.919
25	Euclidean	Distance	.916	.912
30	Euclidean	Distance	.913	.907

It seems that the best F1 value (.94) occurs when number of neighbors are 4 or 5. We proceed with 5 neighbors as the change in F1 value between 3 and 4 neighbors (.940-.938=.002) is greater than the change in F1 value between 4 and 5 neighbors (.940-.939=.001)

b. SVM: while tuning the model, cross-validation with 3-fold yielded the following information:

SVM Type	Cost	E/v	Kernel	Iteration limit	Precision	F1
SVM	1	.1	Linear	100	.853	.852
SVM	1	.1	Polynomial	100	.953	.953
SVM	1	.1	RBF	100	.953	.953
SVM	1	.1	Sigmoid	100	.698	.664
v-SVM	1	.5	Linear	100	.814	.801
v-SVM	1	.5	Polynomial	100	.837	.821
v-SVM	1	.5	RBF	100	.862	.858
v-SVM	1	.5	Sigmoid	100	.777	.768

It seems that the best F1 value occurs when we used SVM with Polynomial and RBF kernel. We could not further tune RBF kernel. Thus, we investigated Polynomial kernel and found the following information:

SVM Type	Cost	E/v	Kernel	Iteration limit	Precision	F1
SVM	1	.1	Polynomial	100	.953	.953
SVM	2	.1	Polynomial	100	.954	.954
SVM	3	.1	Polynomial	100	.954	.954
SVM	4	.1	Polynomial	100	.954	.954
SVM	5	.1	Polynomial	100	.954	.954
SVM	2	1	Polynomial	100	.954	.954
SVM	2	2	Polynomial	100	.954	.954
SVM	2	3	Polynomial	100	.954	.954
SVM	2	4	Polynomial	100	.954	.954
SVM	2	5	Polynomial	100	.954	.954

We further tuned g, c, and d values and found the following data:

SVM Type	Cost	E/v	Kernel	Iteration limit	G	C	D	Precision	F1
SVM	2	.1	Polynomial	100	auto	1	3	.954	.954
SVM	2	.1	Polynomial	100	auto	2	3	.945	.945
SVM	2	.1	Polynomial	100	auto	3	3	.939	.939
SVM	2	.1	Polynomial	100	auto	1	1	.908	.908
SVM	2	.1	Polynomial	100	auto	1	2	.948	.948
SVM	2	.1	Polynomial	100	auto	1	3	.954	.954
SVM	2	.1	Polynomial	100	auto	1	3.5	.954	.954
SVM	2	.1	Polynomial	100	auto	1	3.6	.954	.954
SVM	2	.1	Polynomial	100	auto	1	3.7	.954	.954
SVM	2	.1	Polynomial	100	auto	1	3.8	.954	.954
SVM	2	.1	Polynomial	100	auto	1	3.9	.954	.954
SVM	2	.1	Polynomial	100	auto	1	4	.958	.958
SVM	2	.1	Polynomial	100	auto	1	4.1	.958	.958
SVM	2	.1	Polynomial	100	auto	1	4.2	.958	.958
SVM	2	.1	Polynomial	100	auto	1	4.3	.958	.958
SVM	2	.1	Polynomial	100	auto	1	4.4	.958	.958
SVM	2	.1	Polynomial	100	auto	1	4.5	.958	.958
SVM	2	.1	Polynomial	100	auto	1	4.6	.958	.958
SVM	2	.1	Polynomial	100	auto	1	4.7	.958	.958
SVM	2	.1	Polynomial	100	auto	1	4.8	.958	.958
SVM	2	.1	Polynomial	100	auto	1	4.9	.958	.958
SVM	2	.1	Polynomial	100	auto	1	5	.955	.955
SVM	2	.1	Polynomial	100	auto	1	6	.953	.953

SVM	2	.1	Polynomial	100	1	1	4	.875	.839
SVM	2	.1	Polynomial	100	5	1	4	.874	.833
SVM	2	.1	Polynomial	100	10	1	4	.873	.833
SVM	2	.1	Polynomial	100	.1	1	4	.905	.892

It seems that the best F1 value (.95) occurs when g, c, and d values are automatically detected, 1, and 4 respectively.

c. Random Forest: while tuning the model, cross-validation with 3-fold yielded the following information:

Number of Trees	Attributes at split	Replicable Training	Balance class dist.	Depth	Min subset	Precision	F1
1	N/A	N/A	N/A	N/A	N/A	.720	.720
10	N/A	N/A	N/A	N/A	N/A	.898	.897
20	N/A	N/A	N/A	N/A	N/A	.919	.919
30	N/A	N/A	N/A	N/A	N/A	.926	.926
40	N/A	N/A	N/A	N/A	N/A	.932	.932
50	N/A	N/A	N/A	N/A	N/A	.936	.936
100	N/A	N/A	N/A	N/A	N/A	.939	.939
150	N/A	N/A	N/A	N/A	N/A	.942	.942
200	N/A	N/A	N/A	N/A	N/A	.940	.940

Although as the number of trees increase F1 value increases, we are going to stop at 40 trees to avoid high computation volume. However, we will further tune the other parameters:

Number of Trees	Attributes at split	Replicable Training	Balance class dist.	Depth	Min subset	Precision	F1
40	N/A	N/A	N/A	N/A	N/A	.932	.932
40	10	N/A	N/A	N/A	N/A	.931	.931
40	20	N/A	N/A	N/A	N/A	.931	.931
40	30	N/A	N/A	N/A	N/A	.932	.932
40	40	N/A	N/A	N/A	N/A	.934	.934
40	50	N/A	N/A	N/A	N/A	.930	.930
40	40	Yes	N/A	N/A	N/A	.933	.933
40	40	N/A	Yes	N/A	N/A	.932	.932
40	40	N/A	N/A	10	N/A	.928	.928
40	40	N/A	N/A	20	N/A	.932	.932
40	40	N/A	N/A	30	N/A	.932	.932
40	40	N/A	N/A	30	2	.928	.928
40	40	N/A	N/A	N/A	5	.932	.932
40	40	N/A	N/A	N/A	10	.927	.927

Although there is no significant difference, it seems that the best F1 value (.93) occurs when we have 40 trees, the number of attributes considered at each split is 40, training is not replicable, class distribution is not balanced, and growth is not controlled.

d. Gradient boosting: while tuning the model, cross-validation with 3-fold yielded the following information:

Method	Number of Trees	Learning Rate	Tree Depth	Min subset	Subsampling	Lambda	Precision	F1
Scikit-learn	10	0.1	3	3	.95	N/A	.830	.827
Scikit-learn	20	0.1	3	3	.95	N/A	.868	.867
Scikit-learn	30	0.1	3	3	.95	N/A	.888	.888
Scikit-learn	40	0.1	3	3	.95	N/A	.898	.898
Scikit-learn	50	0.1	3	3	.95	N/A	.908	.908
Scikit-learn	100	0.1	3	3	.95	N/A	.927	.926
cat boost	10	0.1	3	N/A	.95	7	.704	.694
cat boost	20	0.1	3	N/A	.95	7	.874	.870
cat boost	30	0.1	3	N/A	.95	7	.820	.819
cat boost	40	0.1	3	N/A	.95	7	.844	.843
cat boost	50	0.1	3	N/A	.95	7	.857	.856
cat boost	100	0.1	3	N/A	.95	7	.895	.894

After careful consideration, we decide to continue with the Catboost method. Although Catboost needed 100 trees to reach the same F1 value that Scikit-learn reached with 40 trees, it was significantly faster. We will further tune the other parameters for the Catboost method:

Method	Number of Trees	Learning Rate	Tree Depth	Subsampling	Lambda	Precision	F1

cat boost	100	0.1	3	.95	7	.895	.894
cat boost	100	0.2	3	.95	7	.912	.912
cat boost	100	0.3	3	.95	7	.913	.913
cat boost	100	0.4	3	.95	7	.913	.913
cat boost	100	0.5	3	.95	7	.909	.909
cat boost	100	0.3	3	.95	1	.918	.918
cat boost	100	0.3	3	.95	5	.912	.912
cat boost	100	0.3	3	.95	10	.911	.911
cat boost	100	0.3	3	.95	15	.911	.911
cat boost	100	0.3	3	.95	20	.908	.908
cat boost	100	0.3	5	.95	10	.924	.924
cat boost	100	0.3	5	1	10	.927	.927

It seems that the best F1 value (.92) occurs when we have 100 trees, learning rate is 0.3, tree depth is 5, and fraction of features for each tree is 1.

3. Comparing performance. Which model works the best and why:

   a. All the models do data preprocessing steps such as removing instances with unknown target values, continuizesing categorical variables (with one-hot-encoding), removing empty columns, and imputing missing values with mean values. However, KNN further preprocesses the data by normalizing the data by centering to the mean and scaling to a standard deviation of 1.

   b. We used a cross-validation test with three folds to evaluate the models. Tree and Naïve Bayes algorithm didn't turn an F1 score higher than 83 percent. However, the SVM model with an F1 of 95 percent, KNN with an F1 of 94 percent, Random Forest with an F1 of 93 percent, and Gradient Boosting with an F1 of 92 percent yielded pretty good results. When we staked all the models, we got an F1 of 96 percent. Thus, the best approach here would be a stack of all the models as it produces the best possible results. Following are the test results and confusion matrix for each model.

Model	AUC	CA	F1	Precision	Recall
Gradient Boosting	.99	.92	.92	.92	.92
Random Forest	.99	.93	.93	.93	.93
SVM	.99	.95	.95	.95	.95
KNN	.99	.94	.94	.94	.94
Stack	.99	.96	.96	.96	.96
Tree	.90	.80	.80	.80	.80
Naïve Bayes	.98	.83	.83	.84	.83

4. Conclusion: Based on the F1 values, the best model would be an ensemble of all the models (Stack). However, if the intention is to pick just one model, SVM, KNN, Random Forest, and Gradient Boosting are preferred respectively. As the Tree and Naïve Bayes algorithm results are not even close enough to either of the models mentioned earlier, we would not consider these two as acceptable. However, one should always consider that these results are specific to the handwriting prediction problem, and the best model for this problem is not necessarily a good model for approaching other problems out there.

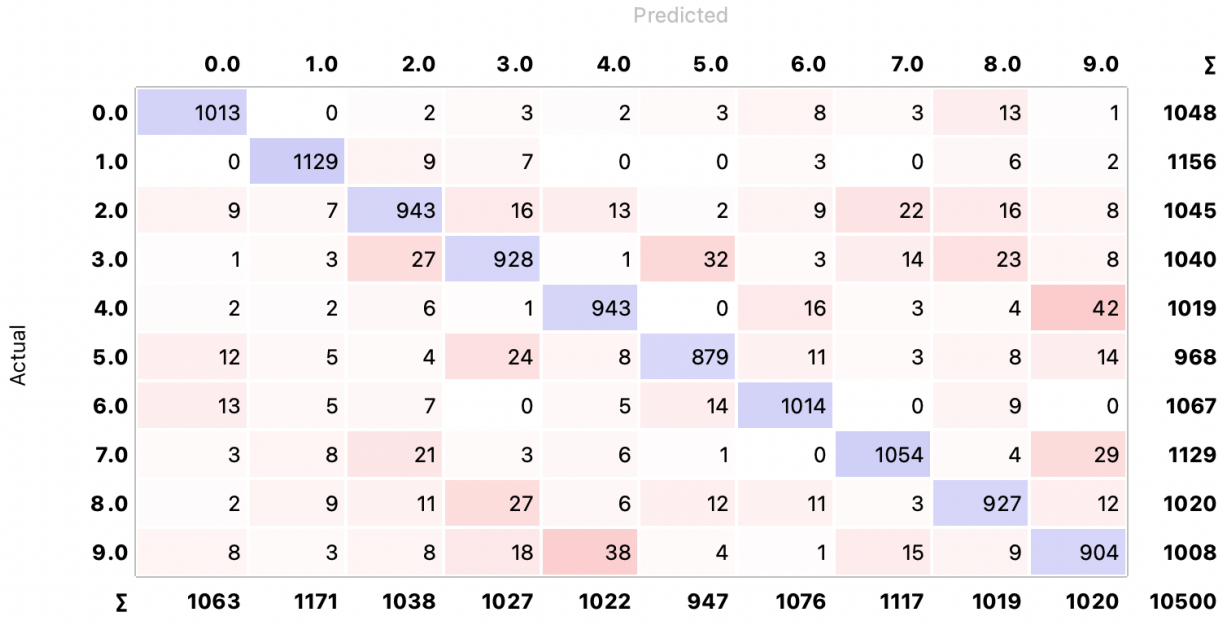

Figure 1: Gradient Boosting Confusion Matrix

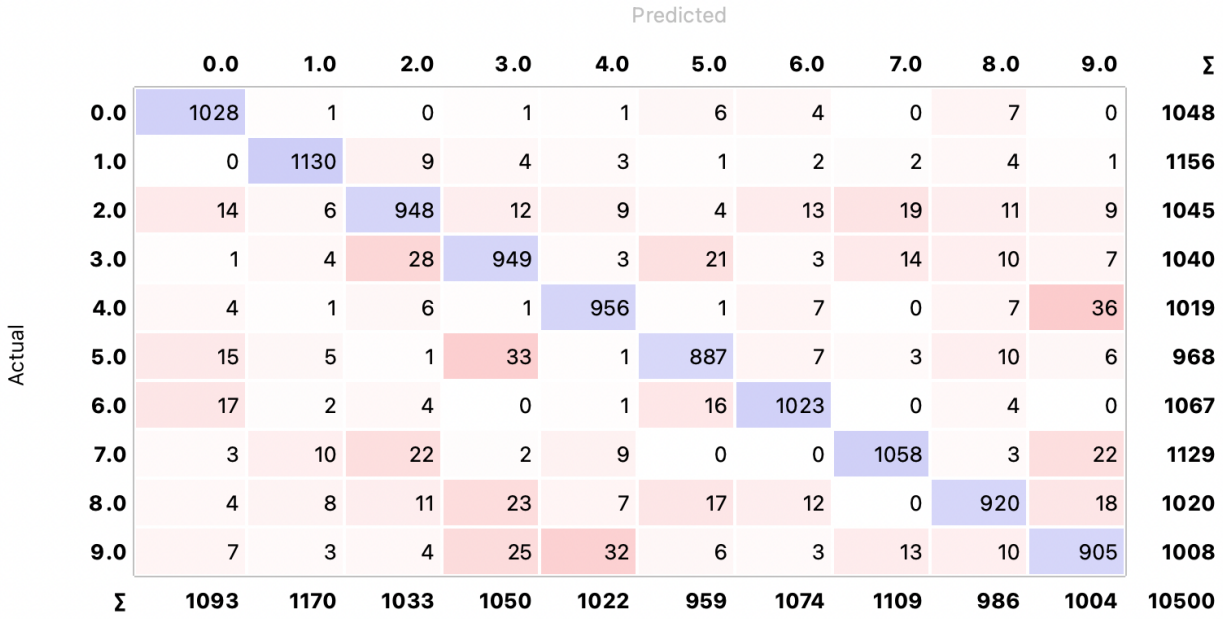

Figure 2: Random Forest Confusion Matrix

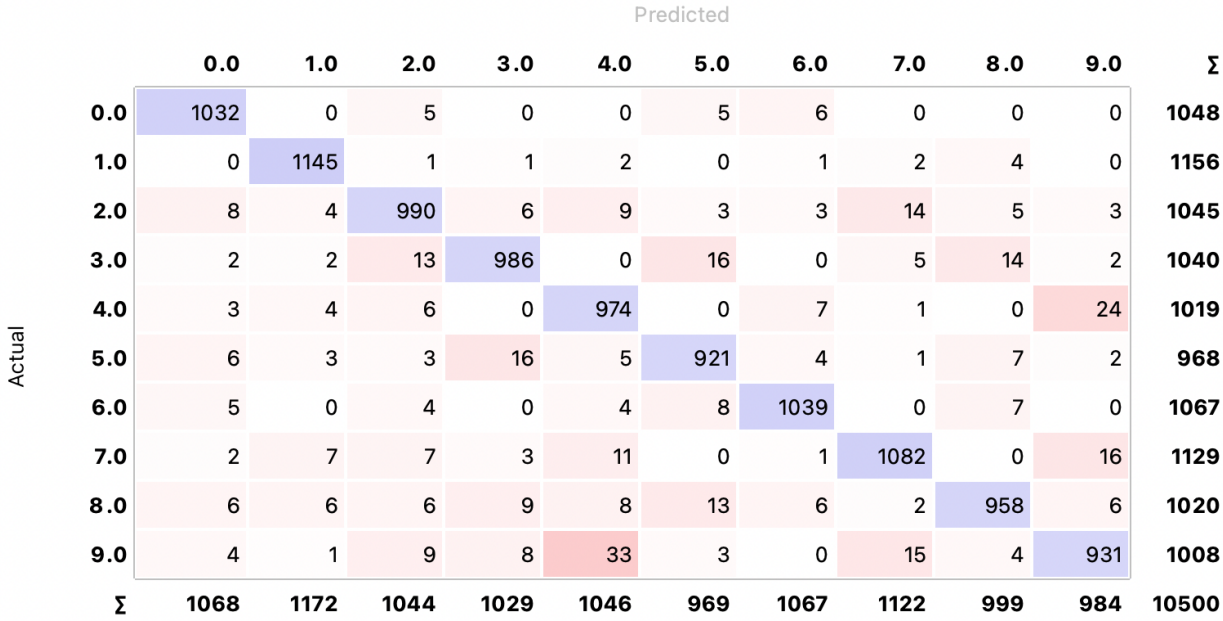

Figure 3: SVM Confusion Matrix

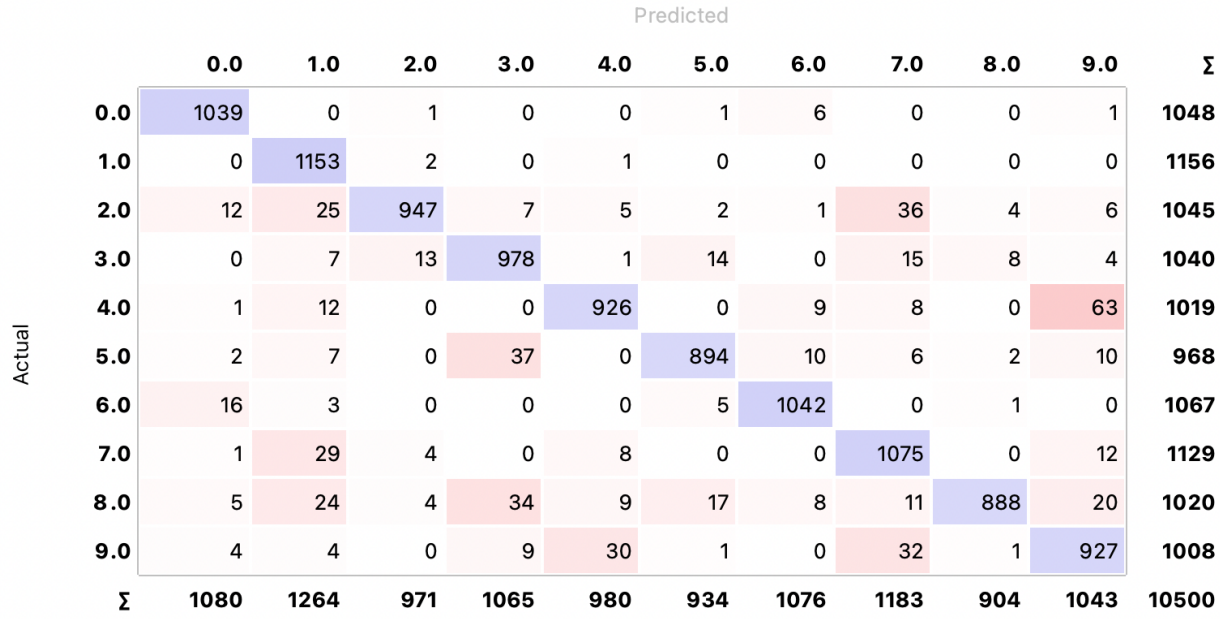

Figure 4: KNN Confusion Matrix

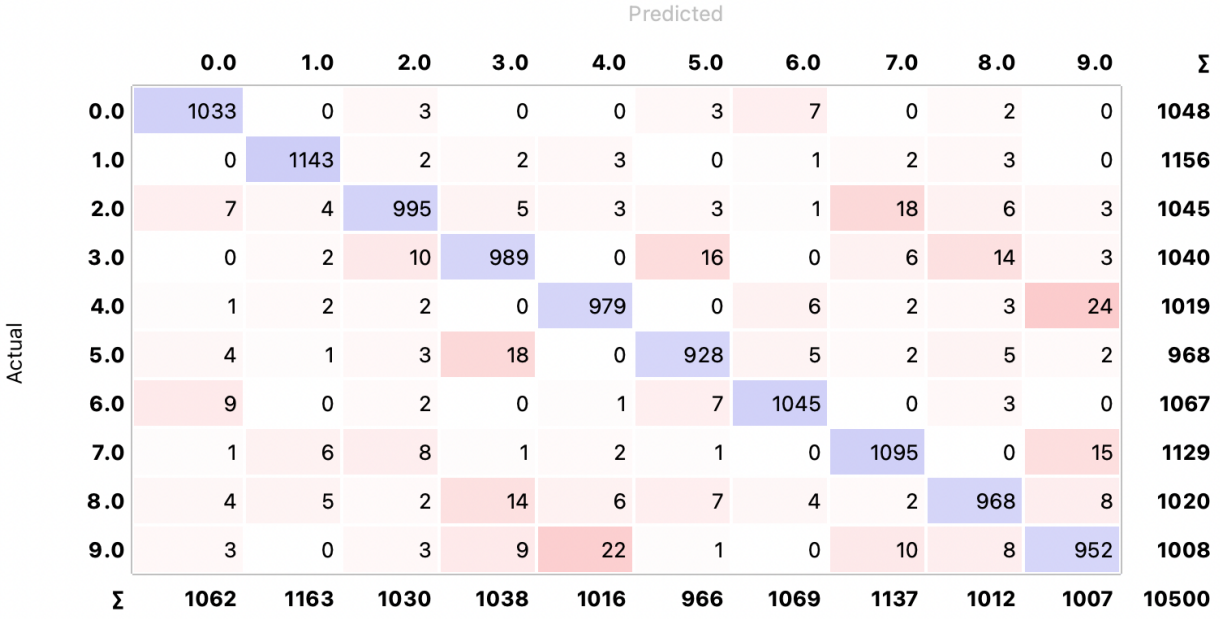

Figure 5: Stack Confusion Matrix