Jacob Dineen

NETID: Jdineen

IST565

Homework 7

**Digit Recognition**

In this experiment we will be looking at a systemically sampled dataset focusing on digit recognition as it relates to different machine learning algorithms. In this case, we will be contrasting success metrics between KNN, SVM and Random Forest algorithms. The purpose of this exercise is to determine which algorithm presents the most accurate model in a controlled setting. The parameter tuning will be slightly different from method to method, including random seed generation, but our testing will be similar (Holdout to Holdout, Cross-Validation to Cross-Validation). With little domain knowledge on digit recognition, I start by looking at the raw data via CSV. I can see that the first column, label, is our class for this dataset, and should be set as such when creating models. The value is nominal/categorically in nature, and represents a value from between 0-9 (10 possible classes) Each row indicates an instance of a digit, so we are looking at 1400 instances. The attributes, or features, are labeled by ‘pixel’ 1..2..3.. etc. After some research, I can see that for this classification problem an image is a 28x28 pixel square, which equals out to 784 pixels. So, it is not a coincidence that we have 784 features within our dataset.

**Problem**

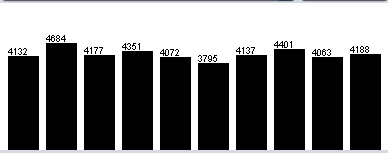
Our classification problem will focus on digit recognition per pixel density, and location, as noted by our numeric columns.

Previously we explored performance metrics using Naive Bayes and Decision Tree (J48) algorithms.

**Input Data**

Each record in our training data includes the nominal categorization of a single digit, as well as the features associated with that digit.

We have a relatively even distribution per our class variable, which will be beneficial for our models.



**Preprocessing**

Because we previously worked within this dataset, there isn’t much need in terms of preprocessing. The label variable needs to be converted from numeric to nominal, and further discretization may take place at the parameter level (We have a very un-normal distribution amongst our pixel variables).

Training will be conducted on the sampled training set (1400 instances) to cut down on our computational load.

Testing will be performed against the larger dataset containing 28000 instances for the purpose of Kaggle submission.

**K Nearest Neighbors (KNN)**

To start the induction process with the KNN algorithm, we navigate to the Lazy folder and select IBk. We know that KNN is considered a lazy algorithm – After further research, we discover that it is lazy because it doesn’t participate in generalization, but rather uses the entire training set during the deduction phase of modeling. What is great about KNN is that it is nonparametric, which means it doesn’t make/rely on assumption about a probability distribution. This is good for our dataset, because we have uneven distributions amongst our features.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Method | Seed | Parameters | Training% |
| KNN | CV. Folds = 5 | 1 | Default | 88.8571% |
| KNN | CV. Folds = 5 | 1 | Weight by 1/distance | 88.8571% |
| KNN | CV. Folds = 5 | 1 | Weight by 1-distance | 88.8571% |
| KNN | Holdout = 66% Train | 1 | Default | 86.7600% |

Our default parameterization was our best model. With few triggers in the KNN algorithm, the distance weighting was adjusted to no change in overall accuracy. Our best model performed at a clip of 88.8571%.

Looking at Fmeasure, we can see that the model had trouble predicting 5’s and 9’s. Venturing down to the confusion matrix, we can see that mistakes in prediction involving the digit 5 were misclassifications into the ‘9’ bucket. Conversely, issues with 9 were the model classifying them as 5. The latter makes more sense to me, lacking domain knowledge in the field, because some people may write their 9’s like their 4’s depending on their style of dictation.

**SVM**

Support Vector Machines are one of the most popular machine learning algorithms around, and were at the forefront before the emergence of artificial and convoluted neural networks. One of the positives about SVMs is that they can use kernel trick to put their inputs/instances into a multidimensional space allowing for a hyperplane to be drawn that wasn’t possible in a two-dimensional space. We ran many controlled tests using the SMO algorithm located in the Functions folder within Weka. Here are the results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Method | Seed | Parameters | Training % |
| SMO | CV. Folds = 5 | 1 | Default | 88.7143% |
| SMO | CV. Folds = 5 | 1 | C = 0.5 | 88.7143% |
| SMO | CV. Folds = 5 | 1 | C = 1.5 | 88.7143% |
| SMO | CV. Folds = 5 | 1 | PolyKernel Exponent = 5 | 82.7100% |
| SMO | CV. Folds = 5 | 1 | Standardize Training Data | 88.8571% |
| SMO | CV. Folds = 5 | 1 | RBF Kernel | 68.7800% |
| SMO | CV. Folds = 5 | 1 | normalized polykernel | 90.0000% |
| SMO | CV. Folds = 5 | 1 | normalized polykernel. Exponent = 5 | 86.2100% |

C is a regularization parameter with the SMO algorithm. When C is larger, we pay a higher cost for misclassification. When large, high chance of overfitting. When small, the margin is wider, and more robust. We noticed that adjusting the C with this dataset didn’t change the results. Rather, we saw changes we changed the Kernel type. Using Crossfold Validation throughout this experiment, we noted our best results using PolyKernel, with an exponent of 5, and normalized PolyKernel. The RBF Kernel, as found through research, is the most popular but garnered the worst results of all models.

Our best model had accuracy of 90%, slightly better than our best KNN model. 2 and 3 were the toughest digits for the model to classify, but there was no pattern in which digit they were misclassified as.

**Random Forest**

Random Forests are one of the more popular machine learning algorithms, and are an extension of Decision trees used in previous assignments. RFs are an example of Ensemble Machine Learning, which means that a bunch of weak learning classifiers can come together and form a strong classifier. The decision trees, depending on the number of iterations, are randomly sampled from the training set with replacement. Majority vote is used to determine prediction results amongst the many trees at the end of the induction process. They are, however, nearly impossible to visualize due to their complexity. We ran many controlled tests using the Random Forest algorithm located in the Trees folder within Weka. Here are the results:

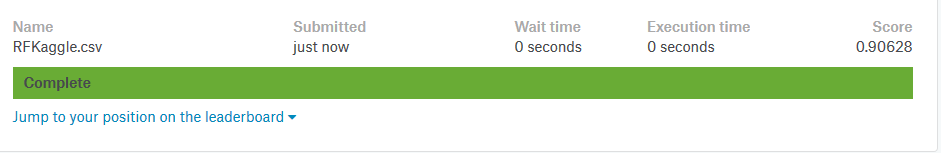
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Method | Seed | Parameters | Training% |
| Random Forest | CV. Folds = 5 | 1 | Default | 90.2143% |
| Random Forest | CV. Folds = 5 | 1 | 50 iterations | 88.2857% |
| Random Forest | CV. Folds = 5 | 1 | 150 iterations | 90.7857% |
| Random Forest | CV. Folds = 5 | 1 | 250 iterations | 90.4200% |

The only significant parameter tuning technique is adjusting the size of the tree, or number of iterations. What was found was that the default setting of 100 iterations performed close to the same as models built using higher complexity/number of trees. Because we were not met with substantially different results, we will utilize our model with default settings to minimize complexity and risk of overfitting the testing data. 9 and 4 were often misclassified as being one another in the predictions

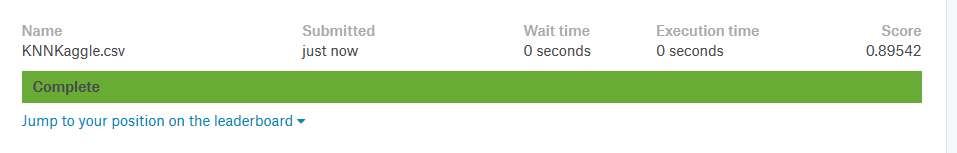
**Kaggle Submissions**

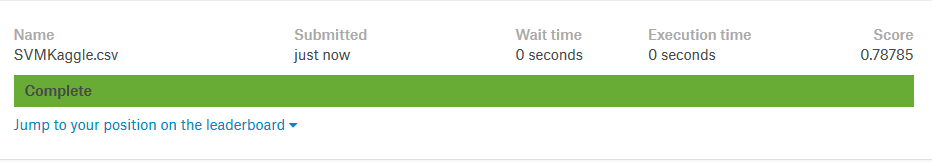
Merge into single dataset with training sample dataset + full testing dataset. Use holdout method with 4.769% used for training. Output the predictions to CSV. We will use parameters from our best models for each algorithm to perform deduction.

Random Forest:



KNN:



SVM 

**Conclusion**

What was found to be very interesting from the application of our induced model onto our testing set was the large amount of overfitting present within the SVM algorithm. It trained at an accuracy of 90%, but tested at a rate of 79%. Our best model with this dataset proved to be generated using the random forest algorithm. My thought is that the Kernel selection may have contributed to the lack of generalization on the training model, or potentially the default weight of misclassifications was not the correct assumption with this dataset.