2019-0703 IST 707 Data Analytics

Homework Assignment 6 (week 7)

Ryan Timbrook
NetID: RTIMBROO
Course: IST 707 Data Analytics
Term: Summer, 2019

Assignment: Naïve Bayes and decision tree for handwriting recognition

Homework Assignment 6 (week 7)

Table of Contents

1 Introduction

Recognize digits 0 to 9 in handwriting images. Use the sampled data to construct prediction models using naive Bayes and decision tree algorithms, additionally tune their parameters to get the best model (measured by cross validation) and compare which algorithms provide better models for this task. The success metric is evaluated on the categorization accuracy of the predictions (the percentage of images predicted correctly).

1.1 About the Data

The data files train.csv and test.csv contain gray-scale images of hand-drawn digits, from zero through nine.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

Each pixel column in the training set has a name like pixelx, where x is an integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed x as x = i * 28 + j, where i and j are integers between 0 and 27, inclusive. Then pixelx is located on row i and column j of a 28 x 28 matrix, (indexing by zero).

For example, pixel31 indicates the pixel that is in the fourth column from the left, and the second row from the top, as in the ascii-diagram below.

Visually, if we omit the "pixel" prefix, the pixels make up the image like this:

```
000 001 002 003 ... 026 027

028 029 030 031 ... 054 055

056 057 058 059 ... 082 083

 |   |   |   |  ...  |   |

728 729 730 731 ... 754 755

756 757 758 759 ... 782 783
```

The test data set, (test.csv), is the same as the training set, except that it does not contain the "label" column.

1.1.1 *Dataset Info*

- Kaggel full training dataset has 42,000 rows and 785 columns
- Kaggel full testing dataset has 28,000 rows and 785 columns
- Kaggel subsampled training dataset has 1,400 rows and 785 columns
- Kaggel subsample testing dataset has 1,000 rows and 785 columns

For this exercise, the full training and testing datasets were used.

1.1.2 *Data Exploration & Cleaning*

There were no NaN fields in the either the kaggel full training or kaggel full testing datasets.

Full training dataset info:

RangeIndex: 42000 entries, 0 to 41999
Columns: 785 entries, label to pixel783
dtypes: int64(785)
memory usage: 251.5 MB

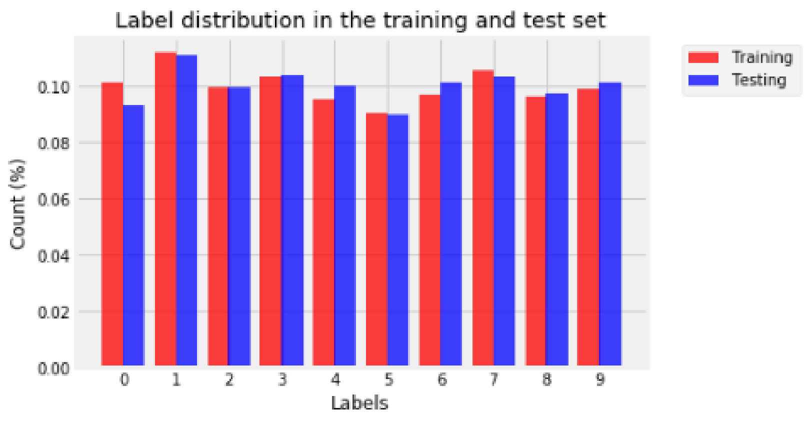
Full testing dataset info:

RangeIndex: 28000 entries, 0 to 27999
Columns: 785 entries, label to pixel783
dtypes: int64(784), object(1)
memory usage: 167.7+ MB

The 'label' attribute was converted to a catagorical, nominal variable for class classification training/testing.

The full training dataset was split into training/testing (70/30 split) sets for model validation and prediction accuracy measurement. The test dataset provided does not have labeled classifications to be used for model validation. It is the unseen data used by the Kaggel competition for submission of predicted classification. In section five, this unseen dataset is used for the competition submission predicitons.

The training/testing datasets created from splitting the full training dataset have nearly equal number of each class label. These datasets are considered balanced.



2 Decision Tree

2.1 Analysis and Models

2.1.1 *Data Preprocessing*

To minimize memory consumption, the datasets attributes were reduced to int32 objects from int64.

2.2 Model - DecisionTreeClassifier

Python package: scikit-learn sklearn.tree.DecisionTreeClassifier

2.2.1 *Model Details*

This DecisionTreeClassifier represents a baseline measurement for the Cross-Validation model detailed in seciont 2.3. The model is fit (trained on) the training dataset described in section 1.1.2 above. (i.e., train_test_split method)

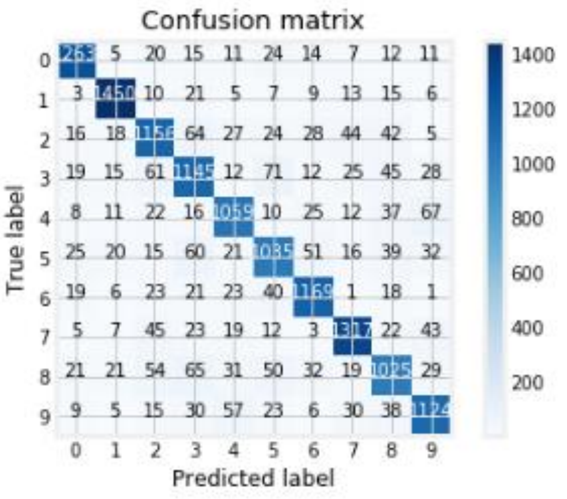
2.2.1.1 Model Parameters

As a baseline model, all default parameters were used in it's build. As such, the default values for the parameters controlling the size of the trees (max_depth, min_samples_leaf) lead to fully grown and unpruned trees; which on this data set lead to very large trees.

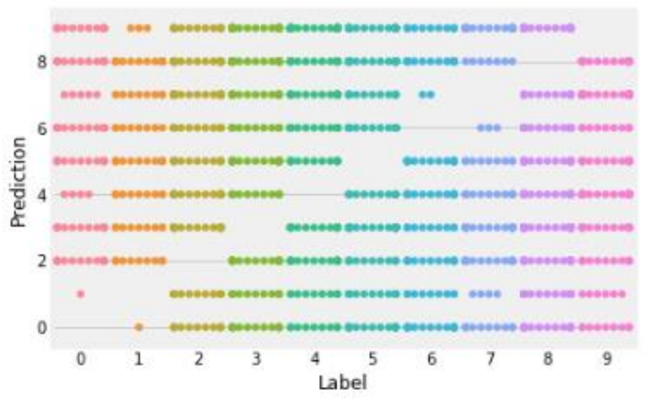
To obtain a deterministic behavior during fitting, random_state was fixed to 0.

2.2.1.2 Model Results

- Fit Score: [0.84726] - 84.7%
- Fit Time: [9.80047]
- Score Time: [0.05957]
- Predict Time: [0.10187]
- Percent Accurately Labeled: [0.84726]



Misclassified Labels:



2.3 Model - DecisionTreeClassifier with Cross-Validation

To avoid the risk of overfitting on a standard test set, cross-validation is used as a means of subdividing the training dataset to hold out a validation set. Training proceeds on the training set, after which evaluation is done on the validation set, when the trials are complete and seem successful, final evaluation is done on the held out test set. In this approach, called k-fold CV, the training set is split into k smaller sets. A model is trainined using k-1 of the folds as training data; the resulting model is validated on the remaining part of the data. (i.e., it is used as a test set to compute a performance measure such as accuracy).

2.3.1 *Model Details*

This is a DecisionTreeClassifier with Cross-Validation. Training and validation are performed on the full training dataset (i.e., without train_test_split methods) using cross-validation procedures. For final evaluation, the held out test set from the train_test_split procedures is used for prediction accuracy measurement. The held out kaggle full test dataset is used for final prediction and competition submission.

2.3.2 *Model Parameters*

The DecisionTreeClassifier object is instantiated with default parameters and random_state equal to 0. Cross validation is used for training and scoring the model with cross validation parameters set to 3. This created three experiments. Overall execution time to process this cross validation procedure with three folds was 33.63 seconds.

Note that cross validation parameters, return_train_score and return_estimator were set to True to report on performance metrics. These parameters increase process execution time by 10 plus seconds.

2.3.3 *Model Results*

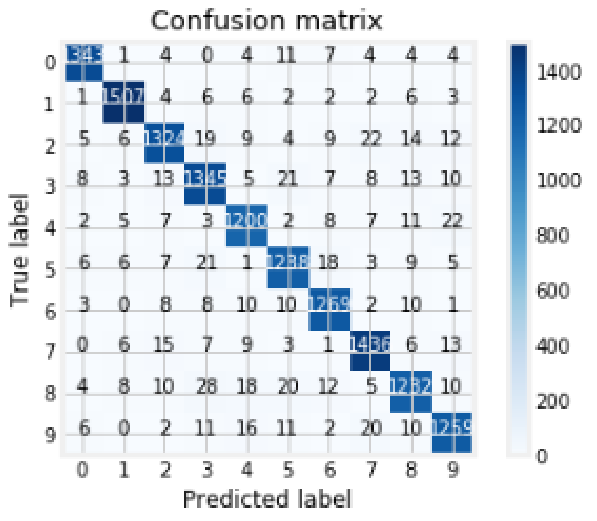
<u>Three fold, Cross Validation Metrics:</u>

Fit Time:	[11.02870226, 10.70894098, 10.53186178]
Score Time:	[0.11198735, 0.10233521, 0.1799221]
Test Recall Scores:	[0.84261269, 0.84340589, 0.84338957]
Test Precision Scores:	[0.84285837, 0.8436598, 0.84341631]
Train Recall Scores:	[1., 1., 1.]
Train Precision Scores:	[1., 1., 1.]

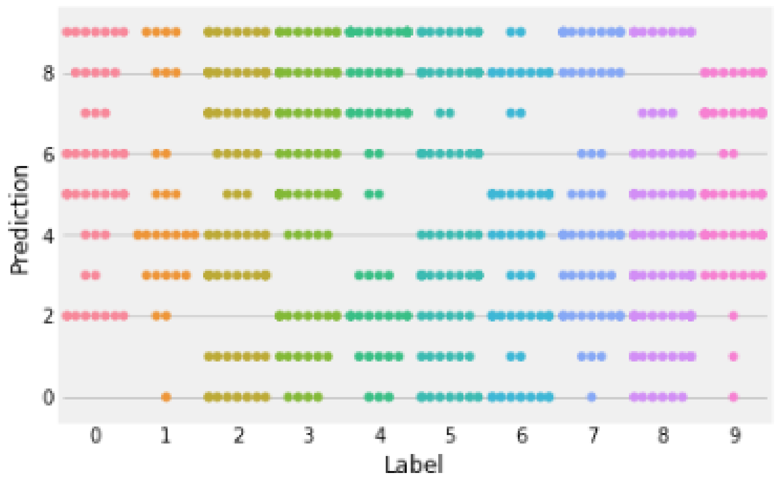
For comparision to the baseline DecisionTreeClassifier, the experment with the best test precision recall was used to predict class labels on the test dataset from the train_test_split procedures.

It's percent accuracy on class labeling was: [0.94898]

*The 10% increase in accuracy is most likely due to overfitting.



<u>Misclassified Labels:</u>



2.4 Model - RandomForestClassifier

Python Package: scikit-learn v0.21.3 sklearn.ensemble.RandomForestClassifier

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default).

2.4.1 *Model Details*

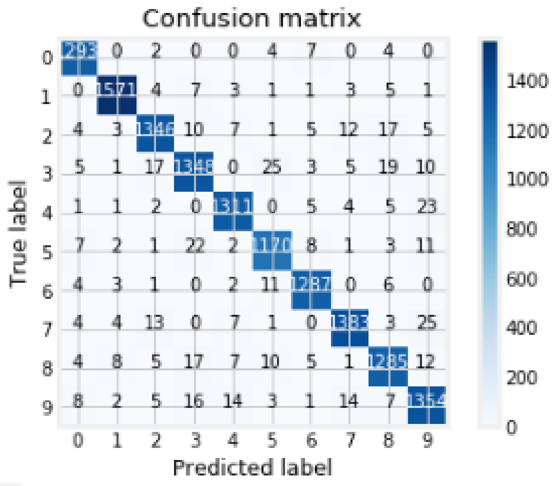
This RandomForestClassifier was include for performance comparison to the above single instance decision tree classifiers. The model was trained on the full training dataset splits from the train_test_split procedures.

2.4.2 *Model Parameters*

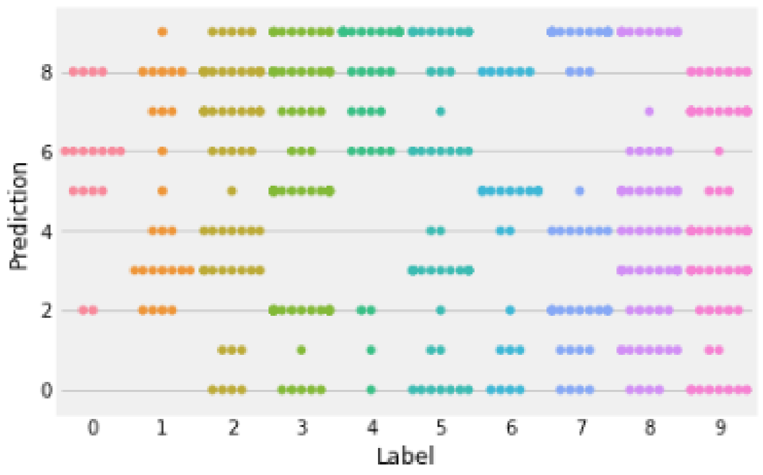
This model was built with n_estimators set to 100 with all other parameters being default.

2.4.3 *Model Results*

- Fit Score: [0.96305] - 96.3%
- Fit Time: [21.0315 s]
- Score Time: [0.85407 s]
- Predict Time: [0.79235 s]
- Percent Accurately Labeled: [0.96305] - 96.3%



Misclassified Labels:



3 Naive Bayes

*T*he **sklearn.naive_bayes** module implements Naive Bayes algorithms. These are supervised learning methods based on applying Bayes' theorem with strong (naive) feature independence assumptions.

3.1 Analysis and Models

3.1.1 *Data Preprocessing*

No additional data preprocessing we done for the Naive Bayes Models.

3.2 Model - Gaussian Naive Bayes Classifier

GaussianNB implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:

3.2.1 *Model Details*

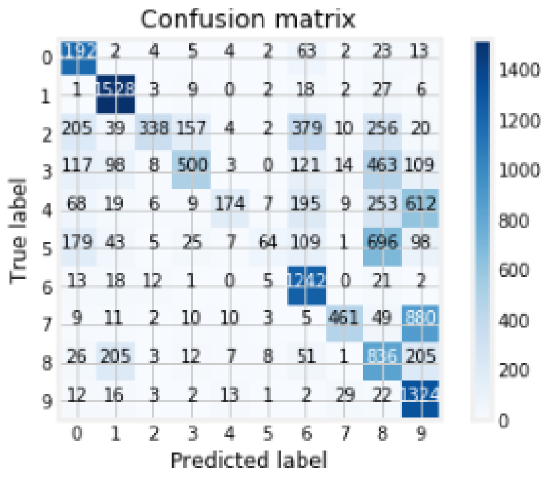
This Gaussian Naive Bayes Classifier represents a baseline measurement for the Cross-Validation model detailed in section 3.4. The model is fit (trained on) the training dataset described in section 1.1.2 above. (i.e., train_test_split method)

3.2.2 *Model Parameters*

As a baseline model, all default parameters were used in it's build.

3.2.3 *Model Results*

- Fit Score: [0.5526] - 55.2%
- Fit Time: [0.5442]
- Score Time: [1.8223]
- Predict Time: [1.7563]
- Percent Accurately Labeled: [0.5525] - 55.2%



	0	1	2	3	4	5	6	7	8	9
0	1192	2	4	5	4	2	63	2	23	13
1	1	1528	3	9	0	2	18	2	27	6
2	205	39	338	157	4	2	379	10	256	20
3	117	98	8	500	3	0	121	14	463	109
4	68	19	6	9	174	7	195	9	253	612
5	179	43	5	25	7	64	109	1	696	98
6	13	18	12	1	0	5	1242	0	21	2
7	9	11	2	10	10	3	5	461	49	880
8	26	205	3	12	7	8	51	1	836	205
9	12	16	3	2	13	1	2	29	22	1324

3.3 Model - Complement Naive Bayes Classifier

The Complement Naive Bayes classifier was designed to correct the “severe assumptions” made by the standard Multinomial Naive Bayes classifier. It is particularly suited for imbalanced data sets.

3.3.1 *Model Details*

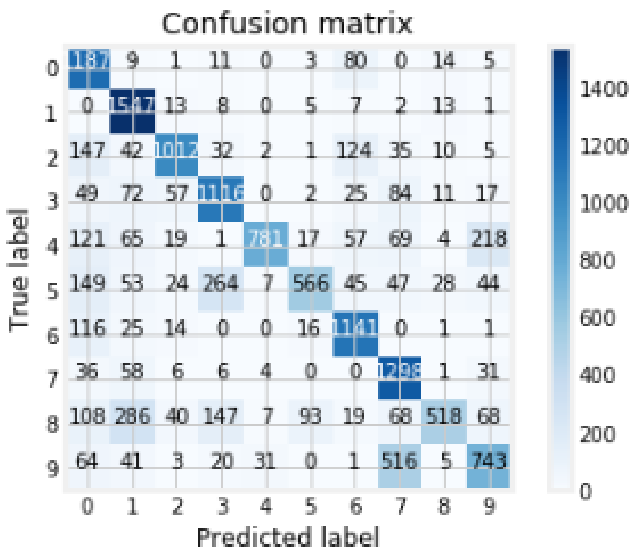
This Complement Naive Bayes Classifier represents a baseline measurement for the Cross-Validation model detailed in section 3.5 as well as a comparison to the Gaussian Naive Bayes Classifier. The model is fit (trained on) the training dataset described in section 1.1.2 above. (i.e., train_test_split method)

3.3.2 *Model Parameters*

As a baseline model, all default parameters were used in it's build.

3.3.3 *Model Results*

- Fit Score: [0.714935] - 71.5%
- Fit Time: [0.192643]
- Score Time: [0.079123]
- Predict Time: [0.076702]
- Percent Accurately Labeled: [0.71493]



3.4 Model - Gaussian Naive Bayes Classifier with CV

3.4.1 *Model Details*

This is a Gaussian Naive Bayes Classifier with Cross-Validation. Training and validation are performed on the full training dataset (i.e., without train_test_split methods) using cross-validation procedures. For final evaluation, the held out test set from the train_test_split procedures is used for prediction accuracy measurement. The held out kaggle full test dataset is used for final prediction and competition submission.

3.4.2 *Model Parameters*

The GaussianNaiveBayes object is instantiated with default parameters. Cross validation is used for training and scoring the model with cross validation parameters set to 3. This created three experiments. Overall execution time to process this cross validation procedure with three folds was 34.4 seconds.

Note that cross validation parameters, return_train_score and return_estimator were set to True to report on performance metrics. These parameters increase process execution time by 10 plus seconds.

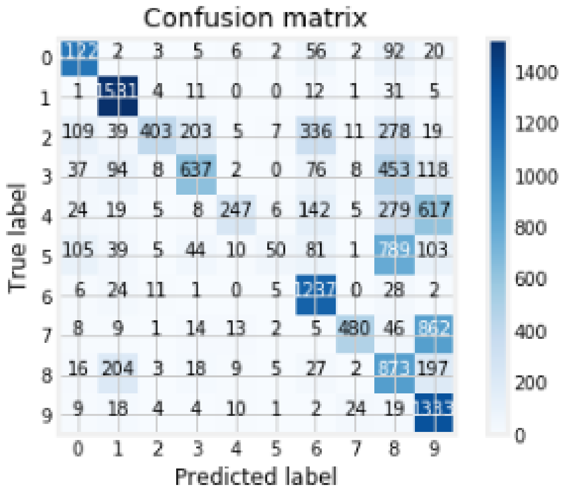
3.4.3 *Model Results*

Three fold, cross validation results:

Fit Time:	[0.90220666, 0.80869913, 0.8218224]
Score Time:	[3.50194836, 3.44823194, 4.44742441]
Test Recall Scores:	[0.55655069, 0.54957274, 0.55724842]
Test Precision Scores:	[0.6740292, 0.66896641, 0.66588789]
Train Recall Scores:	[0.56147661, 0.54797522, 0.56196516]
Train Precision Scores:	[0.68486952, 0.67642211, 0.67696248]

Best Results:

- Fit Score: [0.67402] - 67.4%
- Fit Time: [0.90220]
- Score Time: [3.50194]
- Predict Time: [1.65151]
- Percent Accurately Labeled: [0.57092] - 57%



3.5 Model - Complement Naive Bayes Classifier with CV

3.5.1 *Model Details*

This is a Complement Naive Bayes Classifier with Cross-Validation. Training and validation are performed on the full training dataset (i.e., without train_test_split methods) using cross-validation procedures. For final evaluation, the held out test set from the train_test_split procedures is used for prediction accuracy measurement. The held out kaggle full test dataset is used for final prediction and competition submission.

3.5.2 *Model Parameters*

The ComplementNaiveBayes object is instantiated with default parameters. Cross validation is used for training and scoring the model with cross validation parameters set to 3. This created three experiments. Overall execution time to process this cross validation procedure with three folds was 2.83 seconds.

Note that cross validation parameters, return_train_score and return_estimator were set to True to report on performance metrics. These parameters increase process execution time by 10 plus seconds.

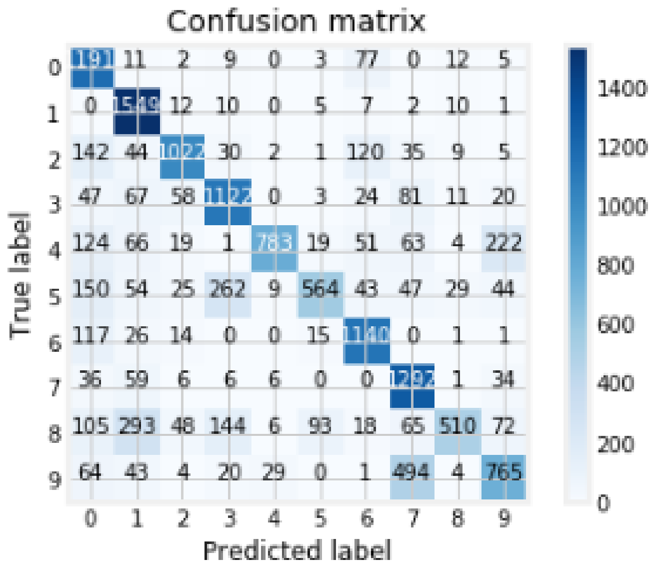
3.5.3 *Model Results*

Three fold, cross validation results:

Fit Time:	[0.48953247, 0.48158693, 0.45195508]
Score Time:	[0.14111781, 0.14356256, 0.15300584]
Test Recall Scores:	[0.71036493, 0.70859042, 0.70432896]
Test Precision Scores:	[0.75009572, 0.74860591, 0.74579701]
Train Recall Scores:	[0.70791525, 0.70864266, 0.71128889]
Train Precision Scores:	[0.74913705, 0.75059161, 0.75032785]

Best Results:

- Fit Score: [0.75009] - 75%
- Fit Time: [0.48953]
- Score Time: [0.14111]
- Predict Time: [0.08291]
- Accurately Labeled: [0.71702] - 71%

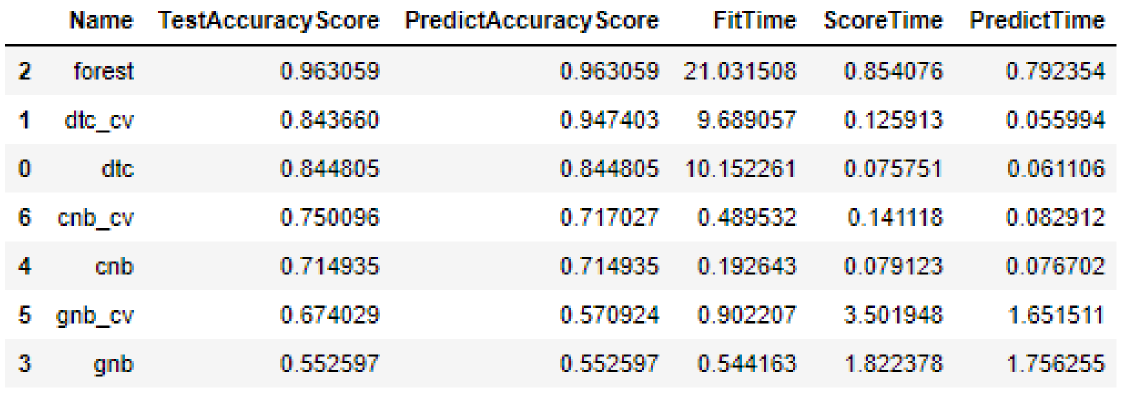


4 Algorithm Performance Comparison

Leaving the random forest algorathm out of this comparison, overall the decission tree algorithms performed better than the naive bayes algorithms, both complement and gaussian implementations, on this data set in predicting hand written digits 0 - 9. The decision tree with three cross fold validation had a 94.7% prediction accuracy to the test dataset and an 84.3% accuracy on the cross-validation test sets. The 10% increase in prediction accuracy over test accuracy is most likely due to overfitting.

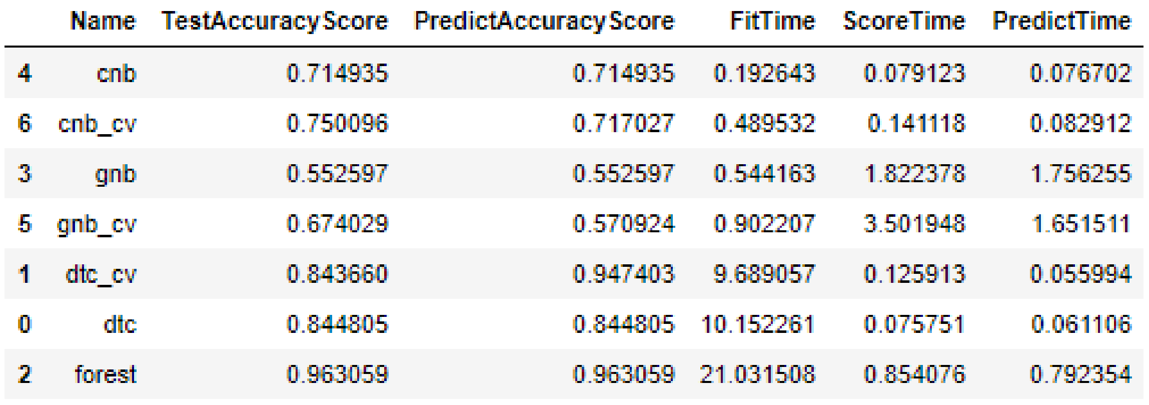
Comparing performance speeds, the naive bayes algorithms out performed the decision tree algorithms substantually. Their speeds were computed in milliseconds, while the decission tree's were in seconds, with the random forest taking the longest at 21 seconds. This would be the result of the number of expermiments produced by the algorithms and how large the tree's could get. Without tuning the tree parameters they can grow very large.

Model Performance Ranking Table, sorted by PredictAccuracyScore descending:



	Name	TestAccuracyScore	PredictAccuracyScore	FitTime	ScoreTime	PredictTime
2	forest	0.963059	0.963059	21.031508	0.854076	0.792354
1	dtc_cv	0.843660	0.947403	9.689057	0.125913	0.055994
0	dtc	0.844805	0.844805	10.152261	0.075751	0.061106
6	cnb_cv	0.750096	0.717027	0.489532	0.141118	0.082912
4	cnb	0.714935	0.714935	0.192643	0.079123	0.076702
5	gnb_cv	0.674029	0.570924	0.902207	3.501948	1.651511
3	gnb	0.552597	0.552597	0.544163	1.822378	1.756255

Model Performance Ranking Table, sorted by FitTime, ScoreTime, PredictTime ascending:



	Name	TestAccuracyScore	PredictAccuracyScore	FitTime	ScoreTime	PredictTime
4	cnb	0.714935	0.714935	0.192643	0.079123	0.076702
6	cnb_cv	0.750096	0.717027	0.489532	0.141118	0.082912
3	gnb	0.552597	0.552597	0.544163	1.822378	1.756255
5	gnb_cv	0.674029	0.570924	0.902207	3.501948	1.651511
1	dtc_cv	0.843660	0.947403	9.689057	0.125913	0.055994
0	dtc	0.844805	0.844805	10.152261	0.075751	0.061106
2	forest	0.963059	0.963059	21.031508	0.854076	0.792354

5 Kaggle Test Result (1 extra point)

5.1 Submission File Format

The submission file should be in the following format: For each of the 28000 images in the test set, output a single line containing the ImageId and the digit predicted. For example, if predict that the first image is of a 3, the second image is of a 7, and the third image is of a 8, then the submission file would look like:

ImageId,Label
1,3
2,7
3,8
(27997 more lines)

The evaluation metric for this contest is the categorization accuracy, or the proportion of test images that are correctly classified. For example, a categorization accuracy of 0.97 indicates that, all but 3% of the images have been correctly classified.

6 Appendix - Grading Rubics

Grading rubrics:

1. Are the models constructed correctly?
2. Is the result analysis conclusion convincing?
3. Is sufficient details provided for others to repeat the analysis?
4. Does the analysis include irrelevant content?
5. Successful submission to Kaggle?