Performance Evaluation of Supervised Learning Algorithms

Govind Rathore, Shivam Mittal, Mohan Zalake, Devansh Soni, Prashant Duhoon

University Of Florida

Abstract

Supervised learning is central to machine learning in aspects of reproducing distinction in data. The ingenuity and utility of different models used to evaluate data can be differentiated based on different criteria and situations offered by certain datasets. A certain learning method that perform well on one criterion may not perform well on other criteria. The undertaken study involves studying and comparing the criterion for some of supervised learning methods i.e. SVMs, Neural Networks, Logistic regression, Naive Bayes, Random forests, CART, Boosted trees and our own Majorized Multiclass kernel SVM. The criterion of comparison includes: Accuracy, Lift, F-Score, Area under the ROC Curve, Average Precision, Precision/Recall Break-Even Point, Squared Error and Cross Entropy. The primary objective of this study is to compare some well established classifiers based on these metrics for popular supervised learning approaches and also compare our results on custom kernelized SVM to a pre-existing SVM library.

1. Introduction

Learning algorithms are now used in various domains and certain performance metrics are appropriate for certain domain. Knowing a model that could predict the true underlying probability for each test case in supervised learning would be optimal. Such an ideal model is called One True Model. We try to optimize our models as close as possible to this one true model, and no other model should yield performance better than it. However, training this models to take into account all underlying probabilities is NP hard. Either correct parametric model type for a domain is not known, or the training sample is small for the model parameters to be estimated accurately; also there is noise in the data usually. In most practical cases, these problems occur together to varying degrees. Even using one true model, it would be hard for its selection from less true models. The performance metrics that will reliably assign best performance to the true model given finite validation data also doesnt exist.

For all practical purposes, we train models to minimize loss measured via a specific performance metric. Since the metrics that could reliably select the one true model doesnt exist, we must select models as close to the optimal solution as possible.

There are many such sub-optimal models and there are different ways that these sub-optimal models can differ from the one true model. Such differences made between different kinds of deviation (sub-optimal models) from the one true model can be reflected in different performance metrics. E.g., ordering metrics such as area under the ROC curve and average precision doesnt make any difference if the predicted values are near the true probabilities,

but depend only on the relative size of the values; i.e. all predictions are scalable; (multiplication/division operation on the values does not change the ROC curve), and metrics based on the ROC curve are also unaffected by such deviations. On the other hand, metrics such as squared error and cross entropy, are affected by scaling the predicted values, but are less affected by small changes in predicted values that might change the relative ordering but not significantly change the deviation from the target values. Similarly, other metrics such as accuracy depend on the predicted values fall with reference to a classification threshold.

2. Methodology

2.1. Learning Algorithms

This section describes the learning algorithms which we have used in this work, their free parameters and their corresponding ranges:

- **K Nearest Neighbor** -An input pattern is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k-nearest neighbors where k is a positive integer,typically small. If k=1, then the object is simply assigned to the class of that single nearest neighbor. We have limited the k to nine values namely 1, 2, 3, 4, 5, 6, 10, 20, 30 due to high amount of processing time requirement. We use kNN with Euclidean distance and uniform weights.

- **LibSVM** We have used libsvm with a fixed radial basis function kernel. The tests were performed by varying C from 1e-4 to 1e+4 by a factor of 10 where C is the penalty for outliers.

- **Random Forest** Random Forest classifies data based on the generation of multiple decision trees using a set features from the training set as split criteria at each of an individual tree and classifies the test data based on the mode of the classes it outputs from the individual tress built using dataset. There are various parameters on which generation of an individual tree depends and some of them are **Number of trees**(n-estimators), **max depth** of an individual tree(max depth), **splitting criterion** at each node, minimum number of samples required to split an internal node and so on.

  For our collection of datasets, we have used n-estimators, max depth and min-samples-split to tune our algorithm. Values for n-estimators varies from 100, 200, 300, 400, 500 for big datasets and 50, 100, 150, 250 for smaller datasets to avoid over-fitting.

  Similarly values of **max-depth varies** from 3 to 8 for every dataset. And values of **min-sample-split** varies from 5,10,15,20,25 samples at each node to decide further splitting of node or not.

- **Naive Bayes** We use naiveBayes function in R as a part of e1071 package to classify our data-sets. It computes the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using the Bayes rule.

$$p(C_k|x) = \frac{p(C_k)p(x|C_k)}{p(x)}$$

- **AdaBoost** Important parameters to tune in adaboost are **base-estimator** ,**n-estimators**, **learning rate**,**algorithm-type** and **individual tuning** parameters of base-estimator For our datasets we have used base-estimator as decision tree, learning rate varies from 0.4,0.6,0.8,1.0,1.2,1.4,1.6, n-Estimators varies from 100,200,300,400 for big datasets and 50,100,150,250 for smaller datasets. Algorithm type used is **SAMME** for discrete prediction and we vary depth of the decision tree from 3 to 8.

- **MultiClass Kernel SVM** It works on the concept finding support vectors from each class which are equidistant from a line/hyper-plane. We are mapping features from k dimension to infinite dimension space using Gaussian kernel. The free parameters will be in this case are C(Penalty for each outlier) and sigma(finding the similarity between two samples).

  For our datasets we have varied value of C from 0.1 to 1000 and sigma values varies are 0.5,1.0,1.5 and 2.0

- **Neural Network** This implementation of Neural Network has only one layer . To find the evidence that an input belongs to a class, each input is multiplied with weights based on the class.

$$evidence_i = \Sigma W_{i,j}\, x_j + b_i$$

  In the above equation, we obtain evidence of whether our input belongs to class I , where $W_{i,j}$ is the weight of an input feature j relative to class i and bi is the class bias. These evidences are then converted to probabilities by means of softmax function.

- **Convolutional Neural Network** A Convolutional Neural Network is a sequence of Convolutional Layers, interspersed with activation functions. In this implementation of Neural Network, a set of filters are convolved over an image. This means sliding the filters over the image spatially, computing dot products. Each dot product can be viewed as a neuron in the network. This is called the convolution layer. Each convolution layer produces a set of activation maps which are stacked together to form a new image. This image is then passed through a pooling layer which is used to make the representations smaller and more manageable. The output of this layer may be fed into the next convolution layer and so on and so forth. After passing through the last pooling layer , the data is fed to the fully connected layer, which contains neurons that connect to the entire input volume, as in ordinary Neural Networks. Finally a softmax layer is applied to determine the class in which the given sample stands greatest possibility to belong.

- **Logistic Regression** Regression model defines distribution of response variables in terms of predictors. Logistic regression models the conditional probability **Pr(Y=1/X=x)**, using maximum likelihood for estimating unknown parameters. It uses logistic transformation log(p/1p), which has an unbounded range. Model for 2 class problem is given by

$$log(p(x))(1 - p(x)) = \beta_0 + x\beta$$

The value of decides the class. The data distribution is considered binomial (for two class) which is easier to interpret and uses link function as

$$log[\pi/(1-\pi)]$$

(logit link),

- **Decision Tree** Cross validation is carried out on training data to decide free parameters (MinLeafSize and MinLeafSize). MinLeafSize and MinLeafSize values which give least k-fold loss are used to train model and classify test data.

2.2. Performance Metric

We have used all eight performance metrics similar to the original paper. These metrics can be sub-categorized into three groups:

Threshold Metrics	**Rank Metrics**	**Probability Metrics**
Accuracy (ACC)	ROC-area (ROC)	Root Mean Squared error (RMS)
F-Score(FSC)	Average Precision (APR)	Cross Entropy (MXE)
Lift(LFT)	Prec-Recall BreakEven point(BEP)	

Table 1: Types of metrics

The threshold metrics do not use the information that how close a prediction is to a threshold, only if the predicted value is above or below threshold. The ordering/rank metrics look at predictions differently from the threshold metrics. If cases are ordered by predicted value, the ordering/rank metrics measure how well the ordering ranks positive cases above negative cases. The rank metrics can be viewed as a summary of the performance of a model across all possible thresholds. Rank metrics depend only on the ordering of the predictions, not the actual predicted values. And at last, the probability metrics depend on the predicted values, not on how the values fall relative to a threshold or relative to each other. The probability metrics are uniquely minimized (in expectation) when the predicted value for each case coincides with the true probability of that case being positive. Below is the description of each metrics:

Accuracy (ACC): Accuracy is the most widely used performance metric in Machine Learning. It is defined as the ratio of machine's correct predictions and total predictions. This metric does not take into account how close/far a prediction is to the true label and assigns a penalty depending on whether the predicted score is above or below a predefined threshold. In our experiments we have used a threshold of 0.5 where predicted score has a range [0, 1].

Precision and Recall: For a given threshold, Precision and Recall are defined as the curve formed by plotting precision vs. recall by varying the threshold. Combining Precision and Recall in different ways gives the metrics such F-score, precision-recall breakeven point and average precision as described next.

$$\text{Precision} = \frac{\text{correctly predicted class memberships}}{\text{predicted class memberships}}$$

$$\text{Recall} = \frac{\text{correctly predicted class memberships}}{\text{true class memberships}}$$

F-score (FSC): For a given threshold, the F-score is the harmonic mean of precision and recall at that threshold. For this work we have used a uniforme threshold of 0.5.
Lift (LFT): Lift is often used in marketing analysis. For a given threshold, it measures how much better a machine is compared to a baseline machine which randomly predicts labels. Usually a fixed percentage of best predicted dataset is used for computing lift. So from the best predicted data, lift gives an estimate of how many are predicted correctly. We have used 25% of dataset for computing lift.

$$\text{Lift} = \frac{\%\text{ of true positives above threshold}}{\%\text{ of dataset above threshold}}$$

ROC-area (ROC): Receiver Operator Characteristics curve is obtained by plotting the true positive rate on the y-axis vs false positive rate on x-axis for a given classifier by varying the threshold of decision boundary. ROC-area is the area under this ROC-curve. The more the ROC-area, the better the classifier. ROC-area is unaffected by calibration and it only depends on the relative ranking of the classes.
Average Precision (APR): It is the area under the precision-recall curve and is mathematically written as,

$$\text{APR} = \int p(r)\, dr$$

which can be closely approximated by,

$$\text{APR} = \sum_{i=1}^{K} P(k)\, \Delta r(k)$$

Where K is the number of thresholds the classifier is validated for. In practice, it is approximated by the average of the precisions at eleven evenly spaced recall levels.
Precision-Recall Breakeven Point (BEP): On the precision-recall curve, it is defined as the value of precision (or recall) when both of them are equal.
Root Mean Square Error (RMS): It measures how much predicted values deviate from the true targets. RMS(error) is defined as,

$$\text{RMS}\ \ \text{error} = \sqrt{\frac{1}{\text{N}} \sum \left(\text{y}_{\text{score}} - \text{y}_{\text{true}}\right)^2}$$

Where y_{score} is the probability of belonging to the class and y_{true} is the true binary class membership $\{0, 1\}$.
Mean Cross Entropy (MXE): Also known as log loss, MXE is used in when one is interested in predicting the probability that an example is positive. MXE is defined as:

$$\text{MXE} = -\frac{1}{\text{N}} \sum \left[\text{y}_{\text{true}} * \ln(\text{y}_{\text{score}}) + (1 - \text{y}_{\text{true}}) * \ln\left(1 - \text{y}_{\text{score}}\right)\right]$$

3. DataSet Description

All Datasets can be described by following table:-

DataSet	#Features	Labels	Size	10%Train	50%Train	90% Test	50% Test
Breast Cancer	11	2	683	68	341	615	342
Optdigits	1024	10	2880	288	4940	2592	1440
Forest Type	27	4	523	52	262	471	261
Car	7	4	1728	172	1364	1556	1364
Nursery	8	4	12960	1296	6480	11664	6480

4. Pre-Processing Data

Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing. Likewise, some of the five datasets used in the project were preprocessed in order to convert them into a format desired for applying the machine learning algorithms.

Forest Mapping Dataset: This dataset has two files training.csv and testing.csv. The first column in the file corresponded to one of the four forest classes . Theses labels were single letters while all other columns were floating point numbers. So the classes were converted from 'd', 'h','s', 'o' to 0,1,2,3 . Moreover, the data from the two files was merged in to a single array so that it can be now split based on training to testing data ratio defined by us.

Cancer Dataset The first column of this dataset corresponded to sample code number which are unique numbers corresponding to each record and do not count as features. So this column is removed from the data during preprocessing

Optical Recognition This dataset consists of images in a 32 x 32 format and corresponding labels of the images. So each 32 x 32 image was flattened to a single record with 1024 input features. The labels corresponding to each image were extracted from the dataset and stored in an array. Additionally, since training and test dataset files were different in this dataset, they were merged to a single file so that we can split the training and test data according to our requirements.

Car Evaluation Each instance in this dataset consisted of 7 columns, first six columns being its input features and the last one being the label. The values in these columns were in string format originally. These values were transformed to consecutive integral values starting from 0. Additionally, the last column was removed stored in a separate array.

Nursery In this dataset we have converted categorical data to numerical data as input to the learning algorithms

In our testing, we tried to standardize all the datasets so that the individual features more or less look like standard normally distributed data: Gaussian with zero mean and unit variance. This did not provide significant performance gain other than in forest dataset. In forest dataset, we saw an improvement in accuracy from around 40% to 85% because of this standardization.

5. Training and Validation

Training We run the algorithms by diving the entire dataset into:

- 10% training, 90% testing
- 50% training, 50% testing

The training set is further divided into training and validation set based on the k value in K-Fold cross validation. We have used 10 fold validation for datasets with high samples and 5 fold for smaller datasets.

K-Fold Cross Validation and Testing In this method, the training data is divided into k groups of instances. Of these, k-1 are used for training the machine, and the k^{th} dataset is used for validation. The K Fold cross validation gives an approximation of how good or how bad the machine performs when we train it on a particular model. We do cross validation for each of our models and the model with the highest mean cross validation accuracy is selected. Now the entire training set(including the validation set) is trained on the selected model and testing is done on the newly trained machine.

6. Multi Class Kernel SVM Algorithm

Problem . Majorisation-based multi-class linear SVM

$$SVM\,Objective\,function : \frac{1}{2}\theta^T\theta + C\sum_{n=1}^{n} max[0, 1 - y_n(\theta^T X_n + \theta_o)]$$

Kernelised SVM:

$$\frac{1}{2}\theta^T\theta + C\sum_{n=1}^{n} max[0, 1 - y_n(\theta^T F_n + \theta_o)]$$

Majorised SVM:

$$\frac{1}{2}\theta^T\theta + C\sum_{n=1}^{n} \frac{[1 - y_n(\theta^T F_n + \theta_o) + z_n]^2}{4z_n}$$

where z= $max(\epsilon, 1 - y(\theta^T F + \theta_o))$
and F=similarity function between features (Gaussian kernel)

Proof. Let $\theta_b = [\theta\, \theta_o]\, and\, F_b = [F\, 1]$

$$= \frac{1}{2}\theta_b^T K\theta_b + C\sum_{n=1}^{n}\frac{[1-y_n(\theta_b^T Fn_b)+z_n]^2}{4z_n}$$

$Differentiating\, w.r.t\, \theta_b$

$$K\theta_b + C\sum_{n=1}^{n} 2\frac{[1-y_n(\theta_b^T Fn_b)+z_n]}{4z_n}(-y_n Fn_b) = 0$$

$$K\theta_b - C\sum_{n=1}^{n}\frac{[1+z_n]}{2z_n}(y_n Fn_b) + C\sum_{n=1}^{n}\frac{[(\theta_b^T Fn_b)Fn_b]}{2z_n} = 0$$

$$K\theta_b + C\sum_{n=1}^{n}\frac{[(Fn_b^T\theta_b)Fn_b]}{2z_n} = C\sum_{n=1}^{n}\frac{[1+z_n]}{2z_n}(y_n Fn_b)$$

$$K\theta_b + C\sum_{n=1}^{n}\frac{[Fn_b^T Fn_b]\theta_b}{2z_n} = C\sum_{n=1}^{n}\frac{[1+z_n]}{2z_n}(y_n Fn_b)$$

$$\theta_b = [K + C\sum_{n=1}^{n}\frac{Fn_b Fn_b^T}{2z_n}]^{-1} C\sum_{n=1}^{n}\frac{[1+z_n]}{2z_n}(y_n Fn_b)$$

$$z = max(|1-y(\theta^T F+\theta_o)|, \epsilon)$$

Above K is the gram kernel matrix which is equivalent to

$$\begin{bmatrix} k(x_1,x_1) & k(x_1,x_2)..... & k(x_1,x_m) \\ k(x_1,x_1) & k(x_1,x_2)..... & k(x_1,x_m) \\ ... & ..... & ..... \\ k(x_m,x_1) & k(x_m,x_2).... & k(x_m,x_m) \end{bmatrix}$$

here m is the number of samples in the dataset and value of each $k(x_i, x_j)$ is :-

$$k(x_i,x_j) = e^{\frac{-C||x_i-x_j||^2}{2\sigma^2}}$$

which is nothing but gaussian kernel. Above the value is being used to update θ_b and z values alternatively until the majorization objective function value converges.

■

Following are the steps to implement the Algorithm:-

1. Load data and divide into training set and testing set.
2. Classify data into one vs all. 1 for sample belonging to class and -1 for any other class.
3. Initialize ϵ,θ_b and z.
4. Features are kernelised using Gaussian function
5. K-fold cross validate the training set to decide C and σ for kernel.
6. Update z, θ_b and loss function E(maj) until it converges.
7. Predict classes for test data using θ_b.
8. Threshold response data at 0 to separate classes into 1 and -1.
9. Evaluate metrics.

7. Observations

Below are the results of our classifier experiments over five data sets. We have compared our classifiers for eight metrics (shown in the columns). The scores in the table represent raw performance metrics which can further be normalized for a better comparison. The first row represents performance metrics of Baseline classifier which unformaly assigns same class label (majority element) to all test vectors. The % data on the title of the table (first cell) represents the percentage of test data left out for generating these tables.

Cancer(50% Test)	**ACC**	**FSC**	**LFT**	**ROC**	**APR**	**BEP**	**RMS**	**MXE**
Baseline	0.651	0.016	1.000	0.504	0.752	0.501	0.591	12.048
Log_reg	0.956	0.937	2.641	0.949	0.951	0.949	0.217	1.584
CART	0.935	0.903	2.722	0.929	0.895	0.895	0.254	2.325
libsvm	0.959	0.945	2.135	0.985	0.976	0.955	0.185	0.121
knn	0.977	0.967	2.195	0.992	0.993	0.974	0.138	0.241
CNN-2 layers	0.985	0.969	1.980	0.998	0.996	0.983	0.102	0.321
Neural Net	0.970	0.940	1.980	0.996	0.991	0.975	0.190	0.879
Random Forest	0.968	0.957	2.230	0.991	0.989	0.956	0.174	0.110
Adaboost	0.956	0.943	2.543	0.986	0.980	0.951	0.326	0.273
MultiClass SVM	0.950	0.967	1.530	0.965	0.955	0.990	0.200	1.470
NaiveBayes	0.660	0.820	2.046	0.755	0.769	0.790	0.441	3.423

Cancer(90% Test)	**ACC**	**FSC**	**LFT**	**ROC**	**APR**	**BEP**	**RMS**	**MXE**
Baseline	0.652	0.009	1.000	0.502	0.751	0.501	0.59	12.035
Log_reg	0.946	0.920	2.721	0.936	0.936	0.941	0.246	1.933
CART	0.917	0.902	2.595	0.932	0.869	0.869	0.265	2.524
libsvm	0.950	0.933	2.065	0.984	0.975	0.943	0.207	0.163
knn	0.967	0.954	2.163	0.969	0.973	0.961	0.180	1.123
CNN-2 layers	0.973	0.961	2.654	0.994	0.992	0.967	0.150	1.234
Neural Net	0.814	0.779	2.341	0.981	0.975	0.945	0.401	1.564
Random Forest	0.966	0.952	2.102	0.990	0.986	0.968	0.185	0.307
Adaboost	0.946	0.924	2.319	0.983	0.983	0.941	0.314	0.213
MultiClass SVM	0.920	0.935	1.528	0.939	0.899	1.000	0.280	2.870
NaiveBayes	0.704	0.794	2.340	0.761	0.771	0.786	0.541	4.024

Forest(50% Test)	**ACC**	**FSC**	**LFT**	**ROC**	**APR**	**BEP**	**RMS**	**MXE**
Baseline	0.382	0.224	1.000	0.510	0.631	0.251	0.556	21.344
Log_reg	0.931	0.862	2.134	0.956	0.923	0.862	0.128	0.690
CART	0.921	0.843	2.195	0.923	0.914	0.843	0.139	0.700
libsvm	0.882	0.882	3.500	0.964	0.890	0.850	0.232	0.428
knn	0.852	0.851	3.457	0.953	0.914	0.854	0.239	1.680
CNN-2 layers	0.842	0.841	2.987	0.953	0.886	0.840	0.244	1.980
Neural Net	0.862	0.861	2.760	0.948	0.883	0.837	0.239	1.640
Random Forest	0.897	0.897	3.178	0.974	0.930	0.892	0.208	0.389
Adaboost	0.900	0.899	2.950	0.980	0.937	0.870	0.417	1.210
MultiClass SVM	0.670	0.280	1.130	0.500	0.800	0.767	0.312	3.270
NaiveBayes	0.855	0.817	2.112	0.876	0.810	0.835	0.351	1.712

Forest(90% Test)	**ACC**	**FSC**	**LFT**	**ROC**	**APR**	**BEP**	**RMS**	**MXE**
Baseline	0.379	0.216	1.000	0.505	0.629	0.251	0.557	21.450
Log_reg	0.845	0.689	1.919	0.809	0.854	0.689	0.198	1.150
CART	0.890	0.781	2.115	0.872	0.887	0.781	0.164	0.987
libsvm	0.830	0.828	3.513	0.953	0.896	0.828	0.255	0.519
knn	0.792	0.792	3.374	0.918	0.854	0.822	0.287	1.847
CNN-2 layers	0.834	0.835	2.987	0.962	0.916	0.854	0.259	4.810
Neural Net	0.878	0.877	2.654	0.968	0.930	0.874	0.221	3.900
Random Forest	0.836	0.835	2.876	0.954	0.893	0.832	0.258	0.714
Adaboost	0.842	0.836	2.971	0.953	0.881	0.835	0.417	1.382
MultiClass SVM	0.650	0.300	1.200	0.500	0.808	0.713	0.190	3.219
NaiveBayes	0.631	0.820	2.040	0.839	0.790	0.791	0.570	2.084

Car(50% Test)	**ACC**	**FSC**	**LFT**	**ROC**	**APR**	**BEP**	**RMS**	**MXE**
Baseline	0.702	0.582	1.000	0.509	0.633	0.251	0.386	10.306
Log_reg	0.927	0.853	2.620	0.940	0.919	0.853	0.087	1.012
CART	0.964	0.928	2.662	0.952	0.958	0.928	0.114	0.469
libsvm	0.976	0.976	3.356	0.473	0.446	0.452	0.187	0.072
knn	0.898	0.892	3.316	0.940	0.858	0.781	0.204	0.778
CNN-2 layers	0.800	0.821	2.820	0.670	0.771	0.758	0.311	0.118
Neural Net	0.773	0.800	2.670	0.650	0.748	0.709	0.253	0.342
Random Forest	0.904	0.892	2.872	0.990	0.897	0.835	0.204	0.938
Adaboost	0.837	0.802	2.765	0.970	0.791	0.706	0.243	1.058
MultiClass SVM	0.967	0.931	2.780	0.880	0.960	0.930	0.100	0.300

Car(90% Test)	**ACC**	**FSC**	**LFT**	**ROC**	**APR**	**BEP**	**RMS**	**MXE**
Baseline	0.701	0.579	1.000	0.505	0.629	0.250	0.387	10.333
Log_reg	0.919	0.761	2.598	0.765	0.912	0.839	0.169	0.342
CART	0.911	0.821	2.634	0.653	0.904	0.821	0.154	0.852
libsvm	0.962	0.962	3.314	0.995	0.987	0.961	0.122	0.122
knn	0.816	0.809	3.308	0.904	0.671	0.687	0.252	1.492
CNN-2 layers	0.766	0.755	2.800	0.877	0.782	0.801	0.151	1.556
Neural Net	0.751	0.702	3.400	0.843	0.759	0.791	0.142	1.721
Random Forest	0.798	0.768	3.065	0.948	0.706	0.657	0.251	1.012
Adaboost	0.804	0.765	2.965	0.972	0.871	0.801	0.244	0.637
MultiClass SVM	0.940	0.880	2.720	0.600	0.930	0.880	0.200	0.300

OptDigits(50% Test)	**ACC**	**FSC**	**LFT**	**ROC**	**APR**	**BEP**	**RMS**	**MXE**
Baseline	0.110	0.032	1.000	0.503	0.553	0.100	0.422	30.725
CART	0.969	0.840	2.832	0.880	0.923	0.845	0.079	0.222
libsvm	0.978	0.978	4.008	1.000	0.997	0.984	0.059	0.765
knn	0.980	0.980	4.008	0.996	0.992	0.982	0.060	0.276
CNN-2 layers	0.972	0.972	3.924	0.999	0.994	0.971	0.067	9.866
Neural Net	0.949	0.949	3.133	0.997	0.987	0.954	0.087	6.654
Random Forest	0.965	0.931	3.732	0.998	0.989	0.960	0.136	0.872
Adaboost	0.968	0.936	2.975	0.998	0.990	0.957	0.114	0.562
NaiveBayes	0.817	0.811	2.090	0.869	0.805	0.830	0.339	2.463

OptDigits(90% Test)	**ACC**	**FSC**	**LFT**	**ROC**	**APR**	**BEP**	**RMS**	**MXE**
Baseline	0.106	0.026	1.000	0.502	0.552	0.100	0.423	30.887
CART	0.940	0.692	2.720	0.743	0.865	0.716	0.129	0.440
libsvm	0.959	0.960	4.005	0.998	0.988	0.962	0.111	0.459
knn	0.932	0.931	4.005	0.992	0.978	0.949	0.102	0.593
CNN-2 layers	0.937	0.937	3.123	0.996	0.977	0.941	0.098	2.360
Neural Net	0.912	0.911	4.232	0.993	0.961	0.918	0.113	3.870
Random Forest	0.939	0.915	3.198	0.996	0.977	0.932	0.168	0.529
Adaboost	0.942	0.923	2.987	0.997	0.965	0.921	0.187	0.497
NaiveBayes	0.832	0.815	2.110	0.872	0.808	0.832	0.345	2.794

Nursery(50% Test)	**ACC**	**FSC**	**LFT**	**ROC**	**APR**	**BEP**	**RMS**	**MXE**
Baseline	0.334	0.168	1.000	0.501	0.626	0.250	0.577	23.010
log_reg	0.881	0.761	1.308	0.754	0.879	0.761	0.085	1.898
CART	0.962	0.916	1.788	0.831	0.952	0.916	0.119	0.512
libsvm	0.997	0.997	3.310	1.000	1.000	0.998	0.035	0.011
knn	0.946	0.945	3.307	0.995	0.976	0.934	0.191	0.302
CNN-2 layers	0.950	0.944	3.269	0.995	0.942	0.895	0.141	0.143
Neural Net	0.765	0.757	3.267	0.918	0.676	0.654	0.273	0.458
Random Forest	0.913	0.902	1.696	0.988	0.945	0.890	0.210	1.256
Adaboost	0.550	0.485	1.986	0.711	0.396	0.349	0.430	1.197
MultiClass SVM	0.721	0.445	1.911	0.632	0.873	0.402	0.265	3.850

Nursery(90% Test)	**ACC**	**FSC**	**LFT**	**ROC**	**APR**	**BEP**	**RMS**	**MXE**
Baseline	0.334	0.167	1.000	0.501	0.625	0.250	0.577	23.017
Log_reg	0.880	0.760	1.336	0.755	0.879	0.760	0.084	1.921
CART	0.944	0.874	1.729	0.889	0.931	0.874	0.146	0.765
libsvm	0.967	0.967	3.308	0.996	0.975	0.934	0.110	0.090
knn	0.883	0.878	3.286	0.965	0.848	0.802	0.254	0.487
CNN-2 layers	0.928	0.923	3.264	0.989	0.905	0.859	0.160	0.179
Neural Net	0.764	0.758	3.261	0.916	0.662	0.646	0.274	0.460
Random Forest	0.898	0.887	1.654	0.982	0.870	0.833	0.216	1.347
Adaboost	0.526	0.485	1.320	0.722	0.414	0.368	0.430	1.302
MultiClass SVM	0.690	0.420	1.890	0.610	0.870	0.350	0.207	4.100

Next two tables present an average performance over all data sets for 50% and 90% test data respectively:

Overall(50% Test)	**ACC**	**FSC**	**LFT**	**ROC**	**APR**	**BEP**	**RMS**	**MXE**
Baseline	0.436	0.204	1.000	0.505	0.639	0.271	0.506	19.487
Log_reg	0.924	0.853	2.176	0.900	0.918	0.856	0.129	1.296
CART	0.950	0.886	2.440	0.903	0.928	0.885	0.141	0.846
libsvm	0.958	0.956	3.262	0.884	0.862	0.848	0.140	0.279
knn	0.931	0.927	3.257	0.975	0.947	0.905	0.166	0.655
CNN-2 layers	0.910	0.909	2.004	0.910	0.909	0.770	0.173	2.486
Neural Net	0.823	0.825	3.019	0.886	0.794	0.762	0.225	1.910
Random Forest	0.929	0.916	2.742	0.988	0.950	0.907	0.186	0.713
Adaboost	0.837	0.813	2.511	0.931	0.822	0.770	0.306	0.881
MultiClass SVM	0.827	0.656	1.838	0.744	0.897	0.772	0.219	2.223
NaiveBayes	0.777	0.816	2.083	0.833	0.795	0.818	0.377	2.533

Overall(90% Test)	**ACC**	**FSC**	**LFT**	**ROC**	**APR**	**BEP**	**RMS**	**MXE**
Baseline	0.434	0.199	1.000	0.503	0.637	0.270	0.507	19.544
Log_reg	0.898	0.783	2.144	0.816	0.895	0.807	0.174	1.336
CART	0.920	0.814	2.359	0.818	0.891	0.812	0.172	1.114
libsvm	0.934	0.930	3.241	0.985	0.964	0.926	0.161	0.271
knn	0.878	0.873	3.227	0.950	0.865	0.844	0.215	1.108
CNN-2 layers	0.888	0.882	2.966	0.964	0.914	0.884	0.164	2.028
Neural Net	0.824	0.805	3.178	0.940	0.857	0.835	0.230	2.303
Random Forest	0.887	0.871	2.579	0.974	0.886	0.844	0.216	0.782
Adaboost	0.812	0.787	2.512	0.925	0.823	0.773	0.318	0.806
MultiClass SVM	0.800	0.634	1.835	0.662	0.877	0.736	0.235	2.622
NaiveBayes	0.722	0.810	2.163	0.824	0.790	0.803	0.485	2.967

Based on the tables populated above, there is no consensus for the selection of the best classifier. A classifier which performs good on on a dataset may not perform the same on other datasets. Our results might have been affected by the fact that some of the dataset have a very less number of feature vectors relative to feature dimensions, e.g. the Forest Type dataset. However, some of the classifiers always perfomed worse that others and some remained close to the top of list irrespective of the nature of dataset. We saw libsvm, CART and convolutional neural networks almost always in top five best performing classifiers. On the other hand, Naive Bayes classifier seems to perform worst almost all of the time.
Our own Majorization based multiclass kernel SVM, is comparable (if not better) to the industry level classifiers for certain datasets such as Breast Cancer and Cars. However, it classifies the Forest Type data set poorly having the worst performance among all other classifiers. Apart from these shortcomings, our SVM is not optimized for speed and takes hours (sometimes days) to compute a single dataset. A few of the table does not have our SVM's data because it is still running. This speed limitations certainly throws it out of competition. Next slowest classifier which we have tested is Adaboost, which takes around 2 hours for OptDigit dataset (still faster than our SVM implementation).

8. Conclusion

On average, for our datasets, libsvm performed best for almost all the metrics. Other classifiers which perfomed well were Random Forest, Convolutional Neural Networks and CART. The models that performed poorest was Naive Bayes. The simple methods such as Logistic Regression and kNN also performed well for our datasets. Although some methods are particularly good for some datasets, a definite conclusion cannot be drawn about the best classifier for a generic problem. Our majorization based kernel multi-class SVM definitly needs a speed-upgrade for it to be a viable alternative of other classifiers. But despite the fact that our SVM is based on simple majorzation steps, it still performs good enough to give us the satisfaction of coding a classifier from scratch.

9. References

1. Rich Caruana,Alexandru Niculescu-Mizil. An Empirical Comparison of Supervised Learning Algorithms. ICML '06 Proceedings of the 23rd international conference on Machine learning
2. Rich Caruana,Niculescu-Mizil2004) Data mining in metric space: An empirical analysis of suppervised learning performance criteria. Knowledge Discovery and Data Mining(KDD 2004)