UNIVERSITY OF FLORIDA

FINAL REPORT

CAP 6610: Machine Learning

Akram Gholami Parekh, Annu Sharma, Venkat Sandeep Katragadda, Kartik Sankara Subramanian and Sai Srivatsav Durgam

4/28/2016

This is the final report for the course CAP 6610: Machine Learning. This report summarizes and provides the findings of the algorithms for SVM, Deep Learning, Random Forests, Kernel SVM (majorized and dual), K-Means clustering and parallelized KNN.

Table of Contents

Preface:

This is a report for the final project for CAP 6610: Machine learning. This report serves a readme for the code submitted as per requirement. The following algorithms have been implemented for as a part of the project.

1. SVM (Matlab)
2. K-Means Clustering
3. Deep Learning
4. Parallelized K-Nearest Neighbors
5. Random Forests
6. Kernel SVM.

The code files for each of the algorithms are present in the folders corresponding to the same name. We have used three datasets as given below:

1. Breast Cancer Wisconsin
2. Optimal Recognition of Handwritten digits.
3. Forest type mapping data

We have performed K-cross validation for validating the free variables. The training data was divided into 10 parts, where all but one was used for training and one for validation, and this was rotated amongst all the 10 parts. The best free parameters were then decided based on the validation results to be used on the test datasets.

Support Vector Machine

With Support Vector Machine we try to find the optimal hyper plane that separates the data points into the required classes. SVM can use a soft margin, meaning a hyperplane that separates many, but not all data points. The standard formulations of soft margin involve adding slack variables ξ_n and a penalty parameter C. In our project, we tried to minimize the $L^1 -$ norm problem. The $L^1 -$ norm problem is:

$$\underset{\theta_l \quad \xi_i}{min} \left(\frac{1}{2} \theta^T \theta + C \sum_j \xi_j \right)$$

Such that:

$$y_n(\theta^T x_n + \theta_0) \geq 1 - \xi_n$$

$$\xi_n \geq 0$$

In these formulation, C is called a *BoxConstraint*, a parameter that controls the maximum penalty imposed on margin-violating observations, and aids in preventing overfitting (regularization). You can see that increasing C places more weight on the slack variables ξ_n, meaning the optimization attempts to make a stricter separation between classes. Equivalently, reducing C towards 0 makes misclassification less important.

fitcecoc, is a command in MATLAB which returns an SVM classifier trained for multi classification, in order to minimize the $L^1 -$ norm problem. The algorithm uses the Lagrange multipliers method to optimize the objective. The three solver options *SMO*, *ISDA*, and *L1QP* for minimization can be used. In our project *SMO*, Sequential Minimal Optimization, which is relatively fast was chosen.

For tuning the SVM classifier this scheme was followed:

1. Pass the data to *fitcecoc*.
2. Cross validate the classifier (10-fold cross validation).
3. Pass the cross-validated SVM model to *kFoldLoss* to estimate and retain the classification error.

4. Retrain the SVM classifier, but adjust the *BoxConstraint* name-value pair arguments. Our strategy is to try a geometric sequence of the *BoxConstraint* parameter. For example, take 40 values, from 10^{-3} to 10^{+1}.

```
% Cross Validation for choosing C
 C = logspace(-3,1,20);

 for c = 1:20

     % SVM command, one versus all
     t = templateSVM('Standardize',1,'BoxConstraint',C(c),'Solver','SMO');

     CVMdl = fitcecoc(Xtrain,Ltrain,'Learners',t,'Coding','onevsall','KFold',10);

     % computing trainingset error
     classLoss = kfoldLoss(CVMdl);

     figure(i)
     semilogx(C(c), classLoss,'b*')
     title('Training set error vs. C ');
     xlabel('C');
     ylabel('Error');
     hold on
 end
```

Choose the model that yields the lowest classification error.

The resulting, trained model (*Mdl*) contains the optimized parameters from the SVM algorithm, enabling us to classify new data.

```
% SVM command, one versus all, choose C by 10-Fold Cross Validation
 t = templateSVM('Standardize',1,'BoxConstraint',C, 'Solver','SMO');
 Mdl = fitcecoc(Xtrain,Ltrain,'Learners',t,'Coding','onevsall');

 % computing testset error
 error = loss(Mdl,Xtest,Ltest);
```

For creating template SVM, if you set *Standardize*, true, MATLAB centers and scales each column of the predictor data (X) by the weighted column mean and standard deviation, respectively. MATLAB trains the classifier using the standardized predictor matrix, but stores the unstandardized data in the classifier property X. It is recommended to try to standardize the predictors. Standardization makes predictors insensitive to the scales on which they are measured.

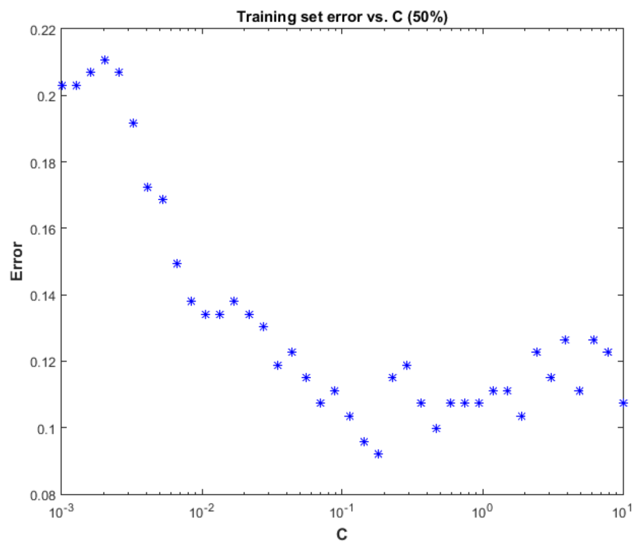
a. **Forest type mapping data**

This data set contains 523 data points with 27 features, from a remote sensing study which mapped different forest types based on their spectral characteristics at visible-to-near infrared wavelengths, using ASTER satellite imagery. This data set is an example of multiclass classification, which consists of four classes, class: *s* ('Sugi' forest), *h* ('Hinoki' forest), *d* ('Mixed deciduous' forest), *o* ('Other' non-forest land).

Multiclass SVM is applied to this data set and the misclassification error is tabulated:

	10%	20%	30%	40%	50%
Training set error	0.0385	0.0481	0.0449	0.0861	0.0690
Test set error	0.1309	0.1321	0.1229	0.1254	0.1301

The test set error is about 13% for $C = 0.1$. By plotting training set error vs. C for every case, it was seen that training set error reaches to a minimum around $C = 0.1$. For instance, training set error vs. C, for 50% of data as training set is illustrated.



The confusion matrix of test set, for 50% of data as training set is:

$$confusion\ matrix\ (50\%) = \begin{bmatrix} 92 & 0 & 4 & 3 \\ 3 & 38 & 0 & 5 \\ 5 & 0 & 34 & 1 \\ 8 & 4 & 1 & 64 \end{bmatrix}$$

$$Class\ order: s, o, h, d$$

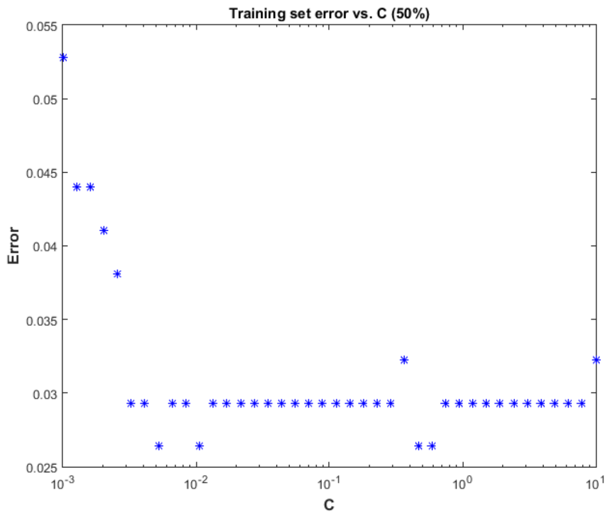
b. Breast Cancer Wisconsin

This data set contains 699 instances with 10 features, from breast cancer databases obtained from the University of Wisconsin. This data set is an example of two class classification, which consists of two classes, Class: 2 for benign, 4 for malignant. There are 16 instances that contain a single missing feature. These points were removed from data set.

Two class SVM is applied to this data set and the misclassification error is tabulated:

	10%	20%	30%	40%	50%
Training set error	0.0441	0.0147	0.0294	0.0330	0.0293
Test set error	0.0490	0.0422	0.0399	0.0375	0.0339

The test set error is about 4% for $C = 0.01$. By plotting training set error vs. C for every case, it was seen that training set error reaches to a minimum around $C = 0.01$. Increasing C might increase training time, so we tried to choose the smallest C. For instance, training set error vs. C, for 50% of data as training set is illustrated.



The confusion matrix of test set, for 50% of data as training set is:

$$confusion\ matrix\ (50\%) = \begin{bmatrix} 214 & 4 \\ 8 & 116 \end{bmatrix}$$

$$Class\ order: 2, 4$$

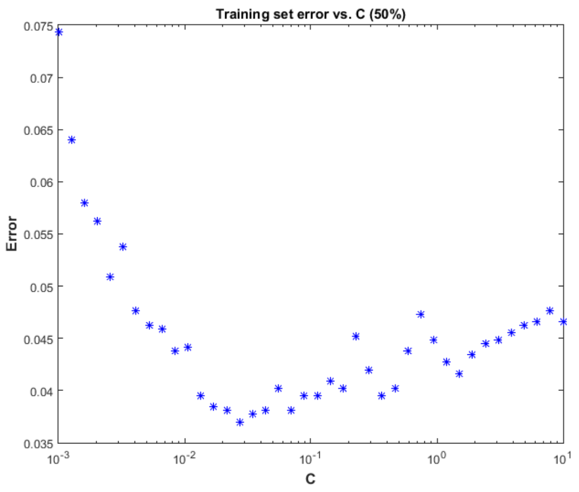
c. Optimal recognition of handwritten digits

This data set contains 5624 data point with 64 features, from a preprocessing programs available by NIST to extract normalized bitmaps of handwritten digits from a preprinted form. This data set is an example of multiclass classification, which consists of ten classes, $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$.

Multiclass SVM is applied to this data set and the misclassification error is tabulated:

	10%	20%	30%	40%	50%
Training set error	0.0125	0.0187	0.0208	0.0196	0.0249
Test set error	0.0566	0.0506	0.0441	0.0396	0.0385

The test set error is about 5% for $C = 0.1$. By plotting training set error vs. C for every case, it was seen that training set error reaches to a minimum around $C = 0.1$. For instance, training set error vs. C, for 50% of data as training set is illustrated.



The confusion matrix of test set, for 50% of data as training set is:

$$confusion\ matrix\ (50\%) =$$

$$\begin{bmatrix} 273 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 279 & 1 & 0 & 0 & 0 & 1 & 2 & 9 & 4 \\ 0 & 0 & 285 & 3 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 263 & 0 & 3 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 268 & 0 & 1 & 0 & 6 & 4 \\ 0 & 0 & 0 & 3 & 0 & 276 & 1 & 0 & 0 & 3 \\ 1 & 0 & 1 & 0 & 3 & 0 & 269 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 281 & 1 & 1 \\ 0 & 7 & 3 & 3 & 2 & 3 & 3 & 0 & 246 & 2 \\ 0 & 4 & 1 & 5 & 3 & 6 & 0 & 0 & 6 & 262 \end{bmatrix}$$

$$Class\ order: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$$

K-Means Clustering

K-means clustering, is an iterative, data-partitioning algorithm that assigns n observations to exactly one of k clusters defined by centroids, where k is chosen before the algorithm starts. K-means treats each observation in the data as an object having a location in space. It finds a partition in which objects within each cluster are as close to each other as possible, and as far from objects in other clusters as possible.

Each cluster in the partition is defined by its member objects and by its centroid, or center. The centroid for each cluster is the point to which the sum of distances from all objects in that cluster is minimized.

The function *kmeans* partitions data into *k* mutually exclusive clusters, and returns the index of the cluster to which it has assigned each observation. By default, *kmeans* uses the *k-means++* algorithm for cluster center initialization and the squared Euclidean metric to determine distances.

$$d(x,c) = (x - c)(x - c)'$$

```
% k-Means command, k is the number of clusters, which must be predefined

[idx,C] = kmeans(X,k,'Distance','sqeuclidean','OnlinePhase','on','Replicates',5);

LCluster = cell(k,1);

% labeling each data point
for i = 1:k

    d            = pdist2(X,C(i,:),'euclidean'); % the distance between data to
                                                   Cluster centroid locations
    [~,ind_min] = min(d);
    LCluster(i) = L(ind_min);

end

Lhat   = LCluster(idx);

% computing error and confusion matrix
kmeans_error = 1-mean(strcmp(Lhat,L))
[D,Order]     = confusionmat(L,Lhat)
```

Like many other types of numerical minimizations, the solution that *kmeans* reaches often depends on the starting points. It is possible for *kmeans* to reach a local minimum, where reassigning any one point to a new cluster would increase the total sum of point-to-centroid distances, but where a better solution does exist. However, you can use the *Replicates* name-value pair argument to overcome that problem. *kmeans* returns the solution with the lowest sum of distances from all objects in the cluster.

If *OnlinePhase* is on, then *kmeans* performs an online update phase (Individually assign observations to a different centroid if the reassignment decreases the sum of the within-cluster, sum-of-squares point-to-cluster-centroid distances.) in addition to a batch update phase (Assign each observation to the cluster with the closest centroid). The online phase can be time consuming for large data sets, but guarantees a solution that is a local minimum of the distance criterion. In other words, the software finds a partition of the data in which moving any single point to a different cluster increases the total sum of distances.

k-means++ algorithm:

The *k-means++ algorithm* uses an heuristic to find centroid seeds for *k*-means clustering. *k*-means++ improves the running time of algorithm, and the quality of the final solution. The *k*-means++ algorithm chooses seeds as follows, assuming the number of clusters is *k*.

1. Select an observation uniformly at random from the data set, *X*. The chosen observation is the first centroid, and is denoted c_1.
2. Compute distances from each observation to c_1. Denote the distance between c_j and the observation *m* as $d(x_m, c_j)$.
3. Select the next centroid c_2, at random from *X* with probability.

$$\frac{d^2(x_m, c_1)}{\sum_{j=1}^{n} d^2(x_j, c_1)}$$

4. To choose center *j*:
    a. Compute the distances from each observation to each centroid, and assign each observation to its closest centroid.

b. For $m = 1, \ldots, n$ and $p = 1, \ldots, j - 1$, select centroid j at random from X with probability

$$\frac{d^2(x_m, c_p)}{\sum_{\{h; x_h \in C_p\}} d^2(x_h, c_p)}$$

where C_p, is the set of all observations closest to centroid c_p and x_m belongs to C_p.

That is, select each subsequent center with a probability proportional to the distance from itself to the closest center that you already chose.

5. Repeat step 4 until k centroids are chosen.

a. Forest type mapping data

The error for this data set is equal to:

$$Error = 0.2218$$

By analyzing the confusion matrix it can be concluded that classes h, and s could be classified better than two other classes. Also it can be seen that a large part of data in class d, were misclassified as class s.

$$confusion\ matrix = \begin{bmatrix} 104 & 44 & 3 & 8 \\ 3 & 166 & 26 & 0 \\ 0 & 4 & 82 & 0 \\ 21 & 6 & 1 & 55 \end{bmatrix}$$

$$Class\ order: d, s, h, o$$

By comparing the misclassification error of SVM algorithm and K-Means for this data set, it is obvious SVM could classify the data much better than K-Means algorithm.

b. Breast Cancer Wisconsin

The error for this data set is:

$$Error = 0.0395$$

By analyzing the confusion matrix it can be seen that class 2 could be classified better than class 1. Also it can be seen that 19 data points in class 4, were misclassified as class 2 and only 9 instances of class 2 was predicted with wrong label.

$$confusion\ matrix = \begin{bmatrix} 435 & 9 \\ 19 & 221 \end{bmatrix}$$

$$Class\ order: 2, 4$$

By comparing the result of SVM algorithm and K-Means, it can be interpreted that SVM and K-Means both had quite the same performance.

c. Optimal recognition of handwritten digits

The error for this data set is equal to:

$$Error\ =\ 0.1961$$

By analyzing the confusion matrix it is obvious K-Means algorithm could not cluster one of the classes (class 9) at all. And most of the data points in that class were misclassified as class 3 and class 1. Also huge part of class 5 was misclassified as class 3.

$$confusion\ matrix = \begin{bmatrix} 549 & 0 & 0 & 0 & 4 & 0 & 1 & 0 & 0 & 0 \\ 0 & 514 & 40 & 10 & 0 & 1 & 4 & 2 & 0 & 0 \\ 1 & 5 & 480 & 29 & 0 & 0 & 1 & 6 & 35 & 0 \\ 0 & 6 & 5 & 508 & 0 & 5 & 0 & 17 & 31 & 0 \\ 0 & 31 & 0 & 0 & 484 & 4 & 4 & 37 & 8 & 0 \\ 0 & 6 & 0 & 114 & 1 & 434 & 3 & 0 & 0 & 0 \\ 1 & 6 & 0 & 0 & 1 & 0 & 549 & 0 & 1 & 0 \\ 0 & 11 & 0 & 0 & 1 & 3 & 0 & 545 & 6 & 0 \\ 0 & 66 & 2 & 22 & 0 & 6 & 5 & 1 & 452 & 0 \\ 0 & 105 & 0 & 406 & 1 & 7 & 0 & 38 & 5 & 0 \end{bmatrix}$$

$$Class\ order: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$$

By comparing the misclassification error of SVM algorithm and K-Means, it is clear SVM could classify the data much better than K-Means algorithm.

Deep Learning:

Deep learning is a class of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using multiple processing layers, with complex structures or otherwise, composed of multiple non-linear transformations

Multilayer perceptron:

A multilayer perceptron (MLP) is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate outputs. An MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Except for the input nodes, each node is a neuron (or processing element) with a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training the network. MLP is a modification of the standard linear perceptron and can distinguish data that are not linearly separable.

The multilayer perceptron consists of three or more layers (an input and an output layer with one or more hidden layers) of nonlinearly-activating nodes and is thus considered a deep neural network. Since an MLP is a Fully Connected Network, each node in one layer connects with a certain weight w_{ij} to every node in the following layer. Some people do not include the input layer when counting the number of layers and there is disagreement about whether w_{ij} should be interpreted as the weight from i to j or the other way around.

An MLP with a single hidden layer can be represented graphically as follows:

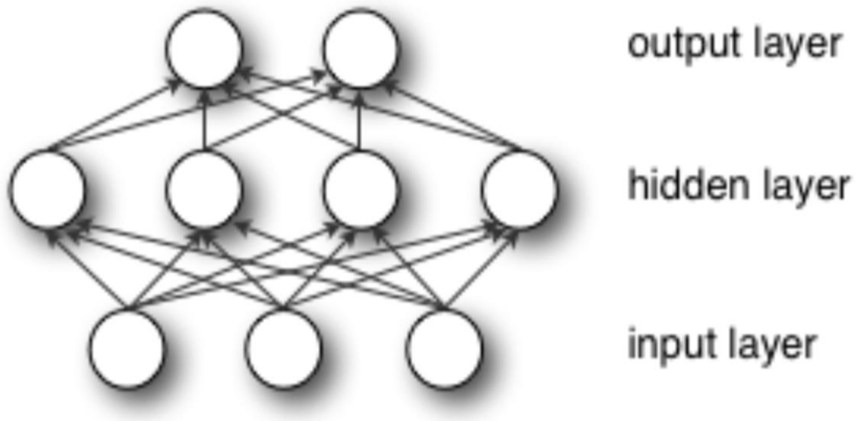


Figure 2. Multi Layer Perceptron

Hence an MLP can be viewed as a logistic regression classifier where the input is first transformed using a learnt non-linear transformation Φ. This transformation projects the input data into a space where it becomes linearly separable. This intermediate layer is referred to as a hidden layer. A single hidden layer is sufficient to make MLPs a universal approximator.

We performed k-fold cross validation by using 20% of the the original training data for validation and the remaining 80% for training the model. This was repeated 5 times by changing the data in the two sets in every trail.

a. Wisconsin Breast Cancer data

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
Accuracy on test data	**93.51%**	**93.21%**	**95.13%**	**94.35%**	**93.51%**

b. Optimal recognition of handwritten digits data

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
Accuracy on test data	**95.89%**	**96.63%**	**95.85%**	**96.13%**	**95.10%**

c. Forest type mapping data

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
Accuracy on test data	**86.42%**	**88.23%**	**87.71%**	**86.17%**	**86.98%**

Other Algorithms implemented:

Logistic Regression:

Logistic Regression is a regression model where the dependent variable is categorical. Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function,

$$f(x) = \frac{L}{1 + \mathrm{e}^{-k(x-x_0)}}$$

. where

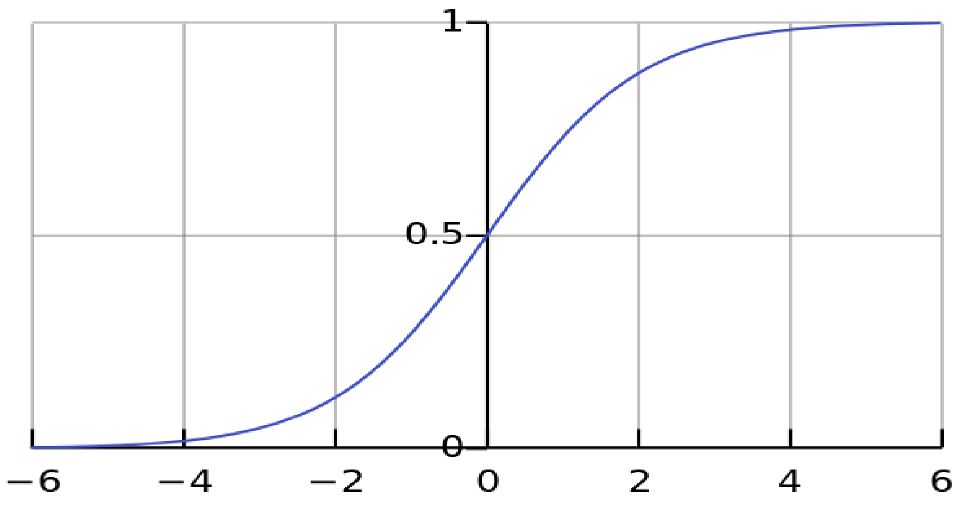


Figure1: Standard logistic sigmoid function

The logistic function is useful because it can take an input with any value from negative to positive infinity, whereas the output always takes values between zero and one and hence is can be interpreted as a probability.

Logistic regression can be binomial or multinomial. Binomial or binary logistic regression deals with situations in which the observed outcome for a dependent variable can have only two possible types. In the project, the Breast cancer Wisconsin (original) dataset involves two target classes and hence is a case of binary logistic regression. Multinomial logistic regression deals with situations where the outcome can have three or more possible types. In the project, the optical recognition of handwritten digits and the the forest type mapping datasets involve more than two target classes and hence we use Multinomial logistic regression for the task of classification.

Learning optimal model parameters involves minimizing a loss function. In the project, we use the negative log-likelihood as the loss. This is equivalent to maximizing the likelihood of the data set D under the model parameterized by θ.

$$\mathcal{L}(\theta = \{W, b\}, \mathcal{D}) = \sum_{i=0}^{|\mathcal{D}|} \log(P(Y = y^{(i)} | x^{(i)}, W, b))$$

$$\ell(\theta = \{W, b\}, \mathcal{D}) = -\mathcal{L}(\theta = \{W, b\}, \mathcal{D})$$

For minimizing the loss function, we use stochastic gradient descent. Stochastic gradient descent (SGD) works according to the same principles as ordinary gradient descent, but proceeds more quickly by estimating the gradient from just a few examples at a time instead of the entire training set.

The experimental results from running logistic regression on the given datasets are as follows. We performed k-fold cross validation by using 20% of the the original training data for validation and the remaining 80% for training the model. This was repeated 5 times by changing the data in the two sets in every trail. The accuracy on the test set is reported,

a. Wisconsin Breast Cancer data

	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**
Accuracy on test data	92.31%	91.37%	92.60%	91.87%	91.66%

b. Optimal recognition of handwritten digits data

	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**
Accuracy on test data	94.59%	93.21%	96.32%	93.47%	95.40%

c. Forest type mapping data

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
Accuracy on test data	83.44%	81.32%	81.93%	82.66%	81.45%

Deep Belief Networks (Deep Learning)

A deep belief network (DBN) is a generative graphical model, or alternatively a type of deep neural network, composed of multiple layers of latent variables ("hidden units"), with connections between the layers but not between units within each layer.

When trained on a set of examples in an unsupervised way, a DBN can learn to probabilistically reconstruct its inputs. The layers then act as feature detectors on inputs. After this learning step, a DBN can be further trained in a supervised way to perform classification.

DBNs can be viewed as a composition of simple, unsupervised networks such as restricted Boltzmann machines (RBMs) where each sub-network's hidden layer serves as the visible layer for the next. This also leads to a fast, layer-by-layer unsupervised training procedure, where contrastive divergence is applied to each sub-network in turn, starting from the "lowest" pair of layers (the lowest visible layer being a training set).

We performed k-fold cross validation by using 20% of the the original training data for validation and the remaining 80% for training the model. This was repeated 5 times by changing the data in the two sets in every trail. The results from running the DBN are as follows,

a.Wisconsin Breast Cancer data

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
Accuracy on test data	63.4 %	67.8%	64.2%	69.3%	68.2%

b. Optimal recognition of handwritten digits data

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
Accuracy on test data	89.5 %	86.1%	86.7%	87.9%	85.4%

c. Forest type mapping data

Accuracy on test data	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
Accuracy on test data	58.3 %	57.2%	57.8%	56.3%	59.7%

It can be observed that the DBN gives good results for the Optimal recognition of handwritten digits data since the dataset is an encoding of an image (high dimensional data) and the network learns good feature representations from the original pixel features. For the other two datasets, the hand-engineered features are good enough for performing classification and since the DBN tries to learn its own representation from this, the performance is not good.

Stacked Autoencoders (Deep Learning)

Denoising autoencoders can be stacked to form a deep network by feeding the latent representation (output code) of the denoising autoencoder found on the layer below as input to the current layer. The unsupervised pre-training of such an architecture is done one layer at a time. Each layer is trained as a denoising autoencoder by minimizing the error in reconstructing its input (which is the output code of the previous layer). Once the first k layers are trained, we can train the k + 1-th layer because we can now compute the code or latent representation from the layer below.

Once all layers are pre-trained, the network goes through a second stage of training called fine-tuning. Here we consider supervised fine-tuning where we want to minimize prediction error on a supervised task. For this, we first add a logistic regression layer on top of the network (more precisely on the output code of the output layer). We then train the entire network as we would train a multilayer perceptron. At this point, we only consider the encoding parts of each auto-encoder. This stage is supervised, since now we use the target class during training.

We can see the stacked denoising autoencoder as having two facades: a list of autoencoders, and an MLP. During pre-training we use the first facade, i.e., we treat our model as a list of autoencoders, and train each autoencoder seperately. In the second stage of training, we use the second facade. These two facades are linked because:

• the autoencoders and the sigmoid layers of the MLP share parameters, and

• the latent representations computed by intermediate layers of the MLP are fed as input to the autoen- coders.

We performed k-fold cross validation by using 20% of the the original training data for validation and the remaining 80% for training the model. This was repeated 5 times by changing the data in the two sets in every trail.

a. Wisconsin Breast Cancer data

	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**
Accuracy on test data	51.21%	55.67%	54.87%	55.33%	51.96%

b. Optimal recognition of handwritten digits data

	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**
Accuracy on test data	73.45%	73.29%	71.66%	74.53%	72.22%

c. Forest type mapping data

	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**
Accuracy on test data	33.51%	32.52%	37.19%	36,54%	33.46%

The stacked autoencoder also tries to learn its own features and the relative performance on the three datasets is similar to that of the deep belief network. Also it can be observed that since the stacked autoencoder is a simpler model without RBMs its accuracy is lower than that of the DBN.

Parallelized K-Nearest Neighbors:

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure. We take an incoming test data point and classify it based on the similarity measure (Euclidian distance in this case) with K most similar points in the training space. As the algorithm involves calculating distances between every test data point with all the training data point, the algorithm has a high time complexity of O(N^2D), where N is the number of data points and D is the number of dimensions of the data.

The parallelized version of the KNN enables us to significantly reduce the running time of the algorithm by parallelizing the distance calculation process. Graphlab (dato) was used to achieve the parallelization. Graphlab is a Python library based on a C++ engine. Once parallelized, the running time of the algorithm is roughly the same as the time taken to get the Euclidian distance between one pair of points, i.e., O(ND).

The function **getDistance()** executes parallely and fetches updates the K nearest neighbors for the given test point. The Euclidean distance is calculated as:

edge['distance'] = np.linalg.norm(np.subtract(src['features'],dst['features']))

$$\text{Distance} = || X_i - X_j ||_2$$

Code Files:

KNN_Digits.py, KNN_Breast_Cancer.py and KNN_Forest_Types.py

Error percentage vs number of nearest neighbors K:

a. Wisconsin Breast Cancer Data:

K	Trail 1	Trail 2	Trail 3	Trail 4	Trail 5
1	4.29%	4.29%	4.53%	4.12%	4.12%
2	7.44%	7.16%	7.16%	7.44%	7.45%
4	5.15%	4.94%	5.08%	5.08%	5.08%
8	3.72%	3.81%	3.92%	3.92%	3.72%
16	4.01%	3.92%	4.01%	4.06%	4.06%

b. Optimal recognition of Handwritten Digits:

K	Trail 1	Trail 2	Trail 3	Trail 4	Trail 5
1	2.01%	2.01%	2.13%	2.01%	2.01%
2	3.11%	3.14%	3.14%	3.19%	3.11%
4	2.32%	2.32%	2.32%	2.35%	2.35%
8	5.04%	4.99%	4.99%	5.04%	4.99%
16	1.63%	1.63%	1.66%	1.70%	1.63%

c. Forest type mapping data:

K	Trail 1	Trail 2	Trail 3	Trail 4	Trail 5
1	13.5%	13.82%	13.11%	13.47%	13.48%
2	15.12%	15.12%	15.52%	15.20%	15.28%
4	16.12%	16.04%	16.32%	16.07%	15.92%
8	19.13%	19.01%	19.52%	19.39%	19.46%
16	17.90%	17.42%	17.63%	17.20%	17.37%

Random Forests

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy. Random decision forests correct for decision trees' habit of overfitting the model to the training set. The algorithm to generate a random forest will create a large number of trees automatically using the concept of bagging, where each tree is based completely on a sample of the given data. The left over data is used for cross validation of this tree's performance.

Once the random forest is generated, the prediction occurs by passing the new input to all the trees present in the forest and classifying the input based on the majority voting. Random forests are categorized under ensemble learning methods. The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator.

Error rates (in %) for various number of trees in forest:

a. Wisconsin Breast Cancer Data:

Trees	**Trail 1**	**Trail 2**	**Trail 3**	**Trail 4**	**Trail 5**
10	2.89855072464	1.44927536232	1.44927536232	1.44927536232	2.89855072464
20	1.44927536232	1.44927536232	1.44927536232	1.44927536232	1.44927536232
30	1.44927536232	1.44927536232	2.89855072464	1.44927536232	1.44927536232
40	1.44927536232	1.44927536232	1.44927536232	1.44927536232	1.44927536232
50	1.44927536232	1.44927536232	1.44927536232	1.44927536232	1.44927536232
60	1.44927536232	1.44927536232	1.44927536232	1.44927536232	1.44927536232
70	1.44927536232	1.44927536232	1.44927536232	1.44927536232	1.44927536232
80	1.44927536232	1.44927536232	1.44927536232	1.44927536232	1.44927536232
90	1.44927536232	1.44927536232	1.44927536232	1.44927536232	1.44927536232

100	1.44927536232	1.44927536232	1.44927536232	1.44927536232	1.44927536232

b. Optimal recognition of Handwritten Digits:

Trees	**Trail 1**	**Trail 2**	**Trail 3**	**Trail 4**	**Trail 5**
10	6.28825820812	5.95436839176	5.17529215359	5.67612687813	6.73344462994
20	4.00667779633	3.61713967724	3.67278797997	4.00667779633	3.9510294936
30	3.56149137451	3.17195325543	3.61713967724	3.89538119087	3.67278797997
40	3.61713967724	3.1163049527	3.45019476906	3.67278797997	3.06065664997
50	3.22760155815	2.78241513634	3.28324986088	3.45019476906	3.50584307179
60	3.39454646633	3.45019476906	3.56149137451	3.33889816361	3.28324986088
70	2.61547022816	3.1163049527	2.72676683361	2.89371174179	2.94936004452
80	3.33889816361	3.39454646633	2.72676683361	2.94936004452	2.89371174179
90	2.83806343907	3.06065664997	2.78241513634	2.94936004452	3.06065664997
100	2.89371174179	2.94936004452	2.89371174179	2.67111853088	3.00500834725

c. Forest type mapping data:

Trees	**Trail 1**	**Trail 2**	**Trail 3**	**Trail 4**	**Trail 5**
10	17.2307692308	16.6153846154	17.2307692308	17.2307692308	19.6923076923
20	18.4615384615	16.9230769231	18.4615384615	18.7692307692	18.1538461538
30	18.4615384615	19.0769230769	18.1538461538	19.0769230769	17.8461538462

40	17.5384615385	16.9230769231	17.5384615385	19.3846153846	19.6923076923
50	19.0769230769	19.6923076923	19.3846153846	19.0769230769	19.6923076923
60	19.6923076923	18.4615384615	19.3846153846	19.0769230769	18.4615384615
70	19.0769230769	19.3846153846	19.0769230769	19.6923076923	18.1538461538
80	19.6923076923	18.1538461538	19.0769230769	19.0769230769	19.3846153846
90	17.8461538462	18.7692307692	18.7692307692	18.4615384615	20.0
100	18.7692307692	18.1538461538	18.7692307692	20.0	18.7692307692

Multi-class Kernel SVM:

As mentioned above, with Support Vector Machines we try to find the optimal hyper plane that separates the data points into the required classes. With the multi-class kernel SVM, we have implemented a kernelized version of the Support vector machine, where we have projected the incoming data points into Recurring Kernel Hilbert Space. For the sake of this project, we have used Gaussian Kernel Hilbert Space.

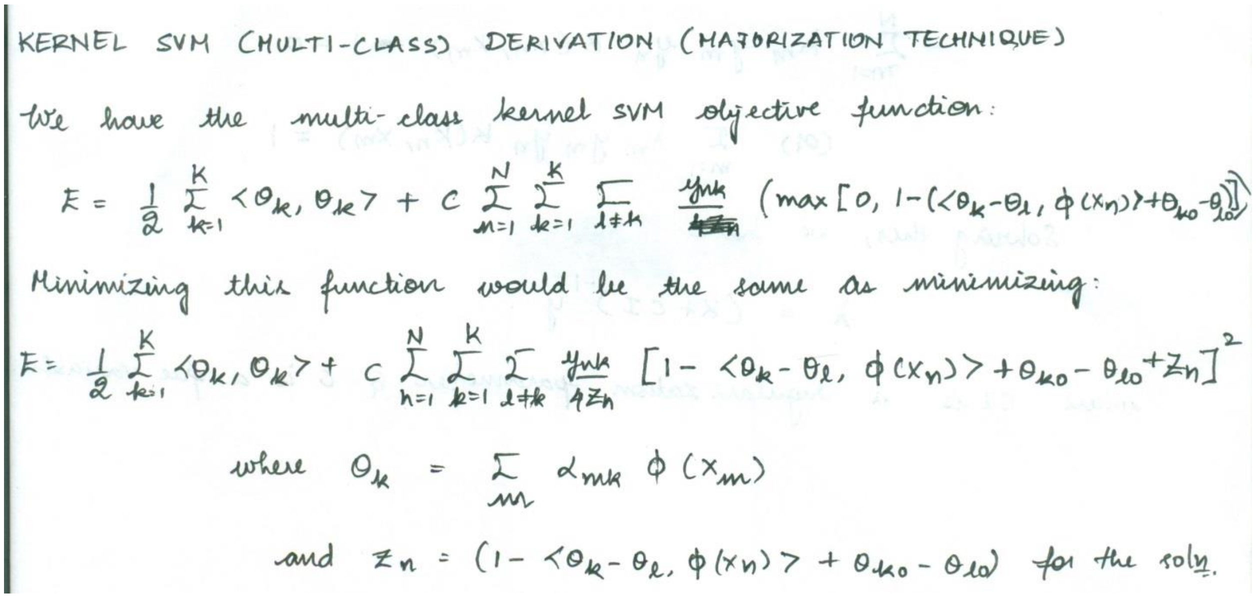
Guassian kernel function: $K(x, y) = \frac{e^{-||x-y||_2^2}}{\sqrt{2\pi}\,\sigma^2}$

We have worked on both the majorization and the dual technique for calculating the classifier. While the dual technique has yielded good results, the majorization technique has not been so fruitful. We have used K-fold cross validation to determine the free variables like C and sigma (Gaussian variance).

Code:

KernelSVM_Dual_Digits.py, KernelSVM_Dual_Breast_Cancer.py, KernelSVM_Maj.py and KernelSVM_Dual_Forest_Types.py

Derivation of the majorization technique:



KERNEL SVM (MULTI-CLASS) DERIVATION (MAJORIZATION TECHNIQUE)

We have the multi-class kernel SVM objective function:

$$E = \frac{1}{2}\sum_{k=1}^{K} \langle \theta_k, \theta_k \rangle + C \sum_{n=1}^{N} \sum_{k=1}^{K} \sum_{l \neq k} \frac{y_{nk}}{\cancel{4z_n}} \left(\max\left[0, 1 - (\langle \theta_k - \theta_l, \phi(x_n) \rangle + \theta_{k0} - \theta_{l0})\right]\right)$$

Minimizing this function would be the same as minimizing:

$$E = \frac{1}{2}\sum_{k=1}^{K} \langle \theta_k, \theta_k \rangle + C \sum_{n=1}^{N} \sum_{k=1}^{K} \sum_{l \neq k} \frac{y_{nk}}{4z_n} \left[1 - \langle \theta_k - \theta_l, \phi(x_n) \rangle + \theta_{k0} - \theta_{l0} + z_n\right]^2$$

where $\theta_k = \sum_m \alpha_{mk} \phi(x_m)$

and $z_n = (1 - \langle \theta_k - \theta_l, \phi(x_n) \rangle + \theta_{k0} - \theta_{l0})$ for the sol$\underline{n}$.

$$E = \frac{1}{2}\sum_{k=1}^{K} \left\langle \sum_{m}^{N} \alpha_{mk}\, \phi(x_m), \sum_{m}^{N} \alpha_{nk}\, \phi(x_n) \right\rangle$$

$$+ C \sum_{n=1}^{N}\sum_{k=1}^{K}\sum_{l \neq k} \frac{y_{nk}}{4 z_n} \left[1 - \left\langle \sum_{m=1}^{N} \alpha_{mk}\, \phi(x_m) - \sum_{n=1}^{N} \alpha_{nl}\, \phi(x_n), \phi(x_n) \right\rangle + \theta_{k0} - \theta_{l0} + z_n \right]^2$$

$$E = \frac{1}{2} \sum_{k=1}^{K} \langle \alpha_k, \alpha_k \rangle \tilde{K} + C \sum_{k=1}^{K}\sum_{l=1}^{K} \frac{\tilde{y}_k}{4} \tilde{Z} \left[-(\alpha_k - \alpha_l) \langle \phi(x_m), \phi(x_n) \rangle + \tilde{\theta} + Z \right]^2$$

where α_k - $n \times 1$ vector of $\alpha_{1k} \dots \alpha_{nk}$

$\tilde{K}$ - kernel matrix $= [\langle \phi(x_m), \phi(x_n) \rangle]$ over all m, n

$\tilde{y}_k$ - $n \times 1$ vector of $y_{1k} \dots y_{nk}$

$\tilde{Z}$ - ~~$n \times 1$~~ $1 \times n$ vector of ~~each~~ elements, $\tilde{Z}_i = \frac{1}{Z_i}$

$\tilde{\theta}$ - $n \times 1$ vector with each element equal to $(1 + \theta_{k0} - \theta_{l0})$

$$E = \frac{1}{2} \alpha_k^T K \alpha + C \sum_{k=1}^{K}\sum_{l=1}^{K} \frac{\tilde{y}_k}{4} \cdot \tilde{Z} \left[\tilde{\theta} - (\alpha_k - \alpha_l) \tilde{K} + Z \right]^2$$

$$\frac{\partial E}{\partial \alpha_k} := \tilde{K} \alpha_k + C \sum_{l \neq k}^{K} \frac{\tilde{y}_k}{4} \tilde{Z}_k \left[\tilde{\theta} - (\alpha_k - \alpha_l)\tilde{K} + Z \right] \cdot (-2\tilde{K}) = 0$$

$$\tilde{K}\alpha_k + C \sum_{l \neq k}^{K} \frac{\tilde{y}_k}{4} \tilde{Z}_k (\tilde{\theta} + Z)(-2\tilde{K}) + C \sum_{l \neq k}^{K} \frac{\tilde{y}_k \tilde{Z}_k}{2} (\alpha_k - \alpha_l) \tilde{K}\tilde{K} = 0$$

(or) $$\tilde{K}\alpha_k + C \sum_{l \neq k}^{K} \frac{\tilde{y}_k \tilde{Z}_k}{2} \alpha_k \tilde{K}\tilde{K} = C \sum_{l \neq k}^{K} \frac{\tilde{y}_k}{2} \tilde{Z}_k (\tilde{\theta} + Z) \tilde{K} + C \sum_{l \neq k}^{K} \frac{\tilde{y}_k \tilde{Z}_k}{2} \alpha_l \tilde{K}\tilde{K}$$

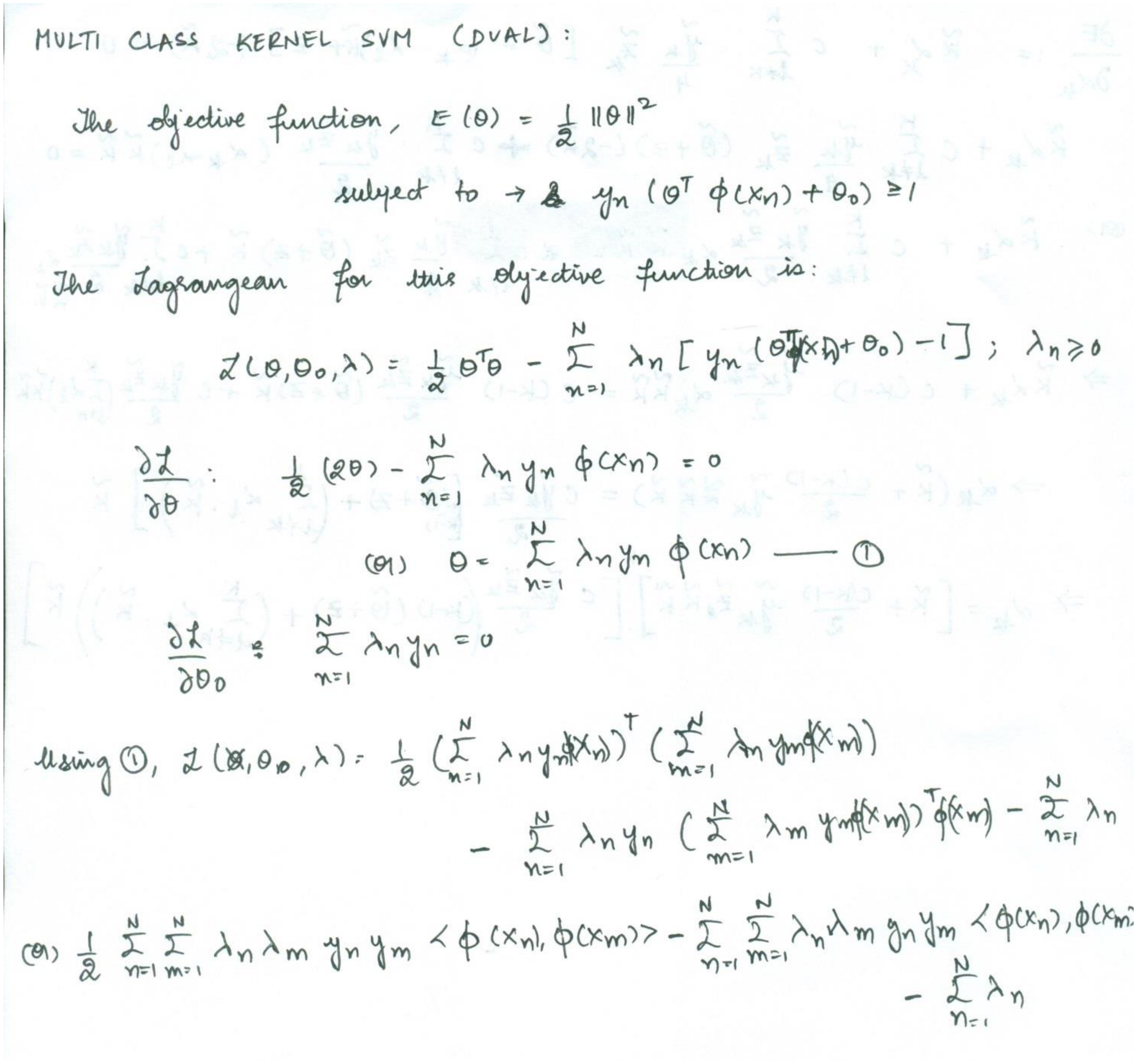
$$\Rightarrow \tilde{K}\alpha_k + C(K-1) \frac{\tilde{y}_k \tilde{Z}_k}{2} \alpha_k \tilde{K}\tilde{K} = C(K-1) \frac{\tilde{y}_k \tilde{Z}_k}{2} (\tilde{\theta} + Z)\tilde{K} + C \frac{\tilde{y}_k \tilde{Z}_k}{2} \left(\sum_{l \neq k}^{K} \alpha_l \right) \tilde{K}$$

$$\Rightarrow \alpha_k \left(\tilde{K} + \frac{C(K-1)}{2} \tilde{y}_k \tilde{Z} \tilde{K}\tilde{K} \right) = \frac{C \tilde{y}_k \tilde{Z}_k}{2} \left[(K-1)(\tilde{\theta} + Z) + \left(\sum_{l \neq k}^{K} \alpha_l \cdot \tilde{K} \right) \right] \tilde{K}$$

$$\Rightarrow \alpha_k = \left[\tilde{K} + \frac{C(K-1)}{2} \tilde{y}_k \tilde{Z}_k \tilde{K}\tilde{K} \right]^{-1} \left[C \frac{\tilde{y}_k \tilde{Z}_k}{2} \left((K-1)(\tilde{\theta} + Z) + \left(\sum_{l \neq k}^{K} \alpha_l \cdot \tilde{K} \right) \right) \tilde{K} \right]$$

We use the equations for α_k and z_k to iteratively update one from the other until the value of α_k converges. We used Python to code these equations to perform majorization on the kernel SVM. We were not able to get good results on this algorithm, with accuracy as low as 20%. The problem that we encountered was that the $\alpha_k$values that were calculated, were very small with variance 10^{-7}. Hence, we also derived and computed the Kernel SVM using the Dual function as well.

Derivation of the dual technique:



MULTI CLASS KERNEL SVM (DUAL):

The objective function, $E(\theta) = \frac{1}{2}\|\theta\|^2$

subject to $\rightarrow$ $y_n(\theta^T \phi(x_n) + \theta_0) \geq 1$

The Lagrangean for this objective function is:

$$\mathcal{L}(\theta, \theta_0, \lambda) = \frac{1}{2}\theta^T\theta - \sum_{n=1}^{N} \lambda_n [y_n(\theta^T\phi(x_n) + \theta_0) - 1] ; \quad \lambda_n \geq 0$$

$$\frac{\partial \mathcal{L}}{\partial \theta} : \quad \frac{1}{2}(2\theta) - \sum_{n=1}^{N} \lambda_n y_n \phi(x_n) = 0$$

$$(\text{or}) \quad \theta = \sum_{n=1}^{N} \lambda_n y_n \phi(x_n) \quad \text{—— } ①$$

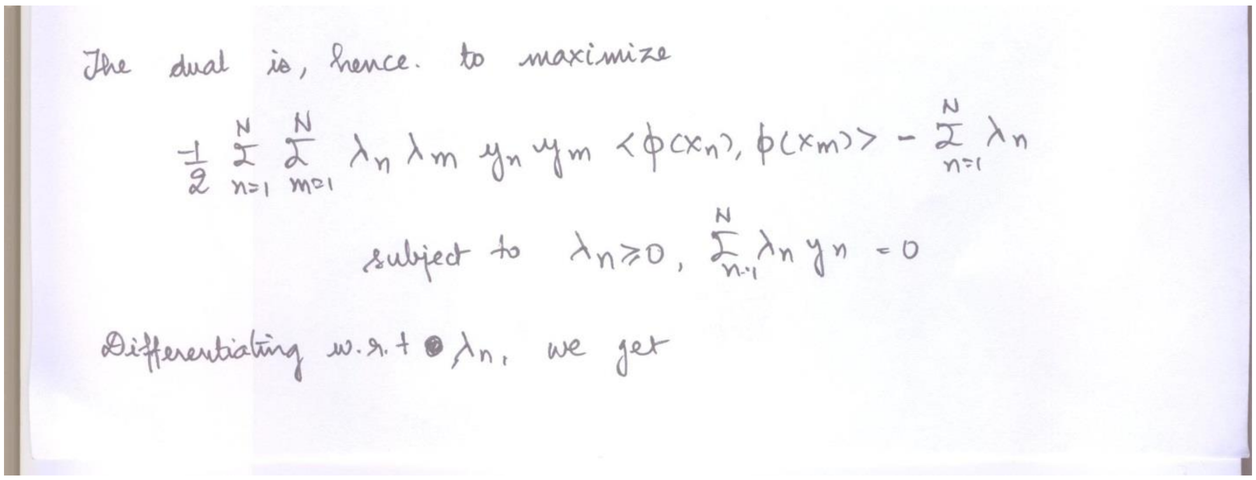
$$\frac{\partial \mathcal{L}}{\partial \theta_0} : \quad \sum_{n=1}^{N} \lambda_n y_n = 0$$

Using ①, $\mathcal{L}(\theta, \theta_0, \lambda) = \frac{1}{2}\left(\sum_{n=1}^{N} \lambda_n y_n \phi(x_n)\right)^T \left(\sum_{m=1}^{N} \lambda_m y_m \phi(x_m)\right)$

$$- \sum_{n=1}^{N} \lambda_n y_n \left(\sum_{m=1}^{N} \lambda_m y_m \phi(x_m)\right)^T \phi(x_m) - \sum_{n=1}^{N} \lambda_n$$

$$(\text{or}) \quad \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m \langle \phi(x_n), \phi(x_m) \rangle - \sum_{n=1}^{N}\sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m \langle \phi(x_n), \phi(x_m$$

$$- \sum_{n=1}^{N} \lambda_n$$

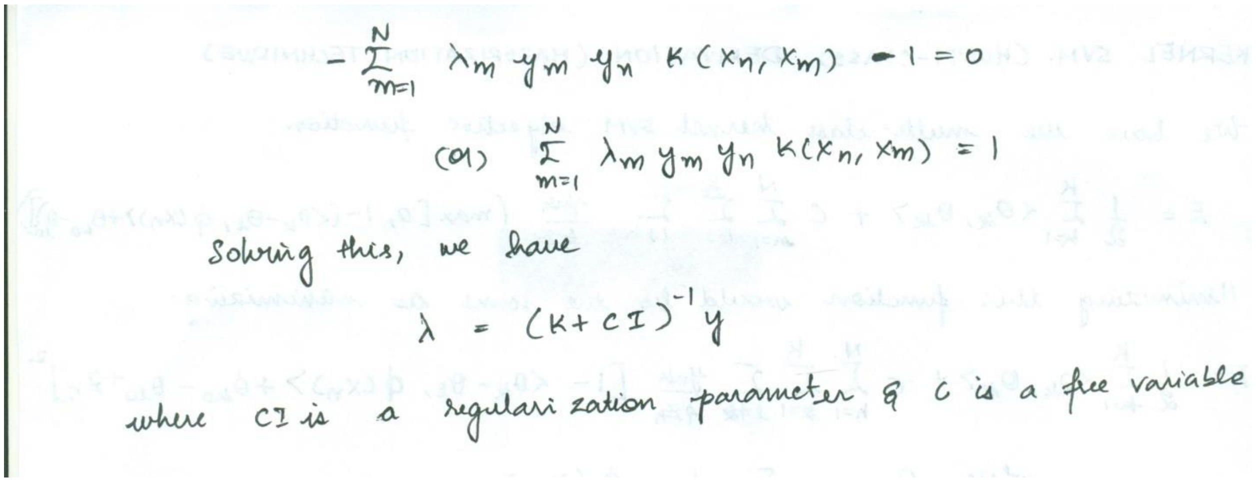


The dual is, hence. to maximize

$$\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}\lambda_n\lambda_m y_n y_m \langle\phi(x_n),\phi(x_m)\rangle - \sum_{n=1}^{N}\lambda_n$$

subject to $\lambda_n \geqslant 0$, $\sum_{n=1}^{N}\lambda_n y_n = 0$

Differentiating w.r.t λ_n, we get



$$-\sum_{m=1}^{N}\lambda_m y_m y_n K(x_n, x_m) - 1 = 0$$

(or) $\sum_{m=1}^{N}\lambda_m y_m y_n K(x_n, x_m) = 1$

Solving this, we have

$$\lambda = (K + CI)^{-1} y$$

where CI is a regulari zation parameter & C is a free variable

Once we have the equation for λ, we use to calculate θ. We use the value of θ to classify the incoming the data points. While this algorithm did very well for the handwritten digits data set, we saw it significantly improve with higher training percentage (80-90%) for the other two. Since the handwritten digits dataset is significantly larger (3823) samples over Breast Cancer (349) and Forest type (189) datasets, it can be concluded that the SVM requires a minimum number of training samples to ensure good results.

Error percentage vs free parameters in the Dual version of Kernel SVM:

a. Wisconsin Breast Cancer Data:

For the validation set:

λ	0.5	1	1.5	2	2.5	3	3.5	4	5
Error	23.12%	22.78%	16.42%	19.22%	15.35%	18.82%	23.24%	26.16%	28.82%

The error on the test data with λ = 2 is 31.13%

b. Optimal recognition of Handwritten Digits:
For the validation set:

λ	0.5	1	1.5	2	2.5	3	3.5	4	5
Error	2.51%	2.18%	0.52%	1.2%	1.65%	1.28%	1.87%	1.32%	2.16%

The error on the test data with λ = 1 is 2.13%

c. Forest type mapping data:
For the validation set:

λ	0.5	1	1.5	2	2.5	3	3.5	4	5
Error	34.16	36.41	36.41	38.23	37.01	31.16	40.32	37.18	35.98

The error on the test data with λ = 1 is 39.13%

Contributions:

SVM and K-Means clustering : Akram Gholami Parekh
Kernel SVM and KNN: Annu Sharma and Sai Srivatsav Durgam
Deep Learning: Kartik Sankara Subramanian and Venkat Sandeep Katragadda
Random Forests: Venkat Sandeep Katragadda