Machine Learning Project

Manu Dixit, Akhil Perincherry,
Francesco Pittaluga, Anand Radhakrishnan,Sowmya Ravi

April 28, 2016

Contents

1 Testing Methodology

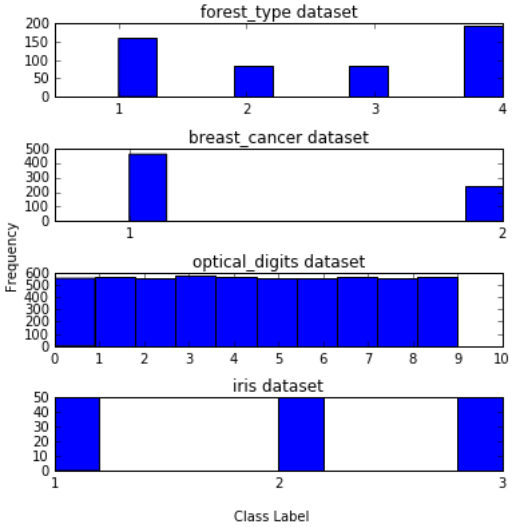
All classifiers were tested on four data sets (*iris,optical digits,forest type and breast cancer*)obtained from the UCI repository. All the data sets were trained and tested with the following training-testing ratios:

Training Set	Test Set
10%	90%
30%	70%
50%	50%

K-fold cross validation was used to set the free parameters for all of the classifiers. The data used for cross validation was taken exclusively from the training set. The size of the dataset, number of features and classes are as follows:

Dataset	Patterns	Features	Classes
Iris	150	4	3
Forest Types	523	27	4
Breast Cancer	699	9	2
Iris	150	4	3

The histograms below show the distribution of different classes in each dataset.



2 Multi-Class Kernel SVM using Majorization

2.1 Derivation of Majorized Objective Function and Update Steps

We have derived a "One Vs All" implementation of a multiclass kernel SVM using majorization. Let us assume that we have K classes and N samples in the training data. Each of the N data samples has d features, i.e. $x_n \in \mathbb{R}^d \ \forall n$ such that $1 \leq n \leq N$. We construct K such machines to separate each class k from every other class l.

The objective function to be minimized is the following:

$$E(\mathbf{\Theta}, \mathbf{\Theta_0}, \{z_n\}) = \frac{1}{2}\sum_{k=1}^{K}\langle\Theta_k, \Theta_k\rangle + C\sum_{n=1}^{N}\sum_{k=1}^{K}\sum_{l\neq k} y_{nk} g(\Theta_k, \Theta_l, \Theta_{k0}, \Theta_{l0}, z_{nlk}) \tag{1}$$

where

$$g(\Theta_k, \Theta_l, \Theta_{k0}, \Theta_{l0}, z_{nlk}) = \frac{1}{4z}[\Delta_{nlk} + z_{nlk}]^2$$

$$\Delta_{nlk} = 1 - (\langle\Theta_k - \Theta_l, \Phi(x_n)\rangle + \Theta_{k0} - \Theta_{l0})$$

$$z_{nlk} = \max\{\epsilon, |\Delta_{nlk}|\}$$

Let $K(.)$ be the kernel function. Then using the property of RKHS

$$K(x_n, x_m) = \langle\Phi(x_m), \Phi(x_n)\rangle \tag{2}$$

From the Representer theorem, we know that the optimal classifier lies within the subspace spanned by the feature vectors (training) in the Hilbert space.

$$\Theta_k = \sum_{m=1}^{N} \alpha_{mk} \Phi(x_m) \tag{3}$$

Substituting Θ_k from equation 3 into equation 1,

$$\begin{aligned} E(\mathbf{\Theta}, \mathbf{\Theta_0}, \{z_n\}) &= \frac{1}{2} \sum_{k=1}^{K} \langle \sum_{m=1}^{N} \alpha_{mk} \Phi(x_m), \sum_{n=1}^{N} \alpha_{nk} \Phi(x_n) \rangle \\ &+ C \sum_{n=1}^{N} \sum_{k=1}^{K} \sum_{l \neq k} \frac{y_{nk}}{4 z_{nlk}} \left(1 - \langle \sum_{m=1}^{N} \alpha_{mk} \Phi(x_m), \Phi(x_n) \rangle + \langle \sum_{m=1}^{N} \alpha_{ml} \Phi(x_m), \Phi(x_n) \rangle - (\Theta_{k0} - \Theta_{l0}) + z_{nlk} \right]^2 \\ &= \frac{1}{2} \sum_{k=1}^{K} \alpha_k^T \tilde{K} \alpha_k + C \sum_{n=1}^{N} \sum_{k=1}^{K} \sum_{l \neq k} \left(\frac{y_{nk}}{4 z_{nlk}} \right) \left(1 - \alpha_k^T K_n + \alpha_l^T K_n - \Theta_{k0} + \Theta_{l0} + z_{nlk} \right)^2 \end{aligned} \tag{4}$$

Here, K_n corresponds to the n^{th} column of the Gram matrix $\tilde{K}$.

$$\tilde{K} = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \dots & \dots & K(x_1, x_N) \\ K(x_2, x_1) & K(x_2, x_2) & \dots & \dots & K(x_2, x_N) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K(x_N, x_1) & K(x_N, x_2) & \dots & \dots & K(x_N, x_N) \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \dots & \dots & \vdots \\ K_1 & K_2 & \dots & \dots & K_N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \dots & \dots & \vdots \end{bmatrix}$$

$$\alpha_k^T = (\alpha_{1k} \ \alpha_{2k} \dots \alpha_{nk} \dots \alpha_{Nk})$$

- **Step 1** : Taking the derivative of equation 4 with respect to α_k

$$\begin{aligned} \frac{\partial E}{\partial \alpha_k} &= \tilde{K} \alpha_k + C \sum_{n=1}^{N} \sum_{l \neq k} \left(\frac{y_{nk}}{2 z_{nlk}} \right) \left(1 - \alpha_k^T K_n + \alpha_l^T K_n - \Theta_{k0} + \Theta_{l0} + z_{nlk} \right) (-K_n) = 0 \\ &= \tilde{K} \alpha_k + C \sum_{n=1}^{N} \sum_{l \neq k} \left(\frac{y_{nk}}{2 z_{nlk}} \right) K_n K_n^T \alpha_k + C \sum_{n=1}^{N} \sum_{l \neq k} \left(\frac{y_{nk}}{2 z_{nlk}} \right) \left(1 + \alpha_l^T K_n - \Theta_{k0} + \Theta_{l0} + z_{nlk} \right) (-K_n) \end{aligned}$$

Therefore,

$$\boxed{\alpha_k = - \left(\tilde{K} + C \sum_{n=1}^{N} \sum_{l \neq k} \left(\frac{y_{nk}}{2 z_{nlk}} \right) K_n K_n^T \right)^{-1} \left(C \sum_{n=1}^{N} \sum_{l \neq k} \left(\frac{y_{nk}}{2 z_{nlk}} \right) \left(1 + \alpha_l^T K_n - \Theta_{k0} + \Theta_{l0} + z_{nlk} \right) (-K_n) \right)} \tag{5}$$

- **Step 2** : Taking the derivative of equation 4 with respect to Θ_{k0},

$$\frac{\partial E}{\partial \Theta_{k0}} = C \sum_{n=1}^{N} \sum_{l \neq k} \left(\frac{y_{nk}}{2 z_{nlk}} \right) \left(1 - \alpha_k^T K_n + \alpha_l^T K_n - \Theta_{k0} + \Theta_{l0} + z_{nlk} \right) (-1) = 0$$

Therefore,

$$\boxed{\Theta_{k0} = \frac{\sum_{n=1}^{N} \sum_{l \neq k} \left(\frac{y_{nk}}{2 z_{nlk}} \right) \left(1 - \alpha_k^T K_n + \alpha_l^T K_n + \Theta_{l0} + z_{nlk} \right)}{\sum_{n=1}^{N} \sum_{l \neq k} \left(\frac{y_{nk}}{2 z_{nlk}} \right)}} \tag{6}$$

- **Step 3**: The value of z_{nlk} that maximizes the majorized function g is given below:

$$\begin{aligned}\Delta_{nlk} &= 1 - (\langle \Theta_k - \Theta_l, \Phi(x_n) \rangle + \Theta_{k0} - \Theta_{l0}) \\ &= 1 - \left(\langle \sum_{m=1}^{N} \alpha_{mk} \Phi(x_m) - \sum_{m=1}^{N} \alpha_{ml} \Phi(x_m), \Phi(x_n) \rangle + \Theta_{k0} - \Theta_{l0} \right) \\ &= 1 - \left(\sum_{m=1}^{N} \alpha_{mk} K(x_m, x_n) - \sum_{m=1}^{N} \alpha_{ml} K(x_m, x_n) + \Theta_{k0} - \Theta_{l0} \right)\end{aligned}$$

Therefore,

$$\boxed{\Delta_{nlk} = 1 - \left(\alpha_k^T K_n - \alpha_l^T K_n + \Theta_{k0} - \Theta_{l0} \right)} \tag{7}$$

$$z_{nlk} = \max\{\epsilon, |\Delta_{nlk}|\}$$

Steps 1, 2 and 3 are run iteratively until a chosen convergence criteria is met. Values α_k^*, Θ_{k0}^* obtained at the end of the iterations are chosen for the K SVMs.

2.2 Testing procedure

To evaluate the performance of the machine on test data samples x_t, each of these K machines are run on the test data and classes are assigned based on the following criteria:

$$\boxed{C_t^* = \arg\max_{k^*} \left(\sum_{n=1}^{N} \alpha_{nk^*} K(x_t, x_n) + \Theta_{k^*0} \right)} \tag{8}$$

2.3 Results and Discussion

All the four datasets were run with no kernel (linear SVM) and a Gaussian kernel. We tried the experiment for three different training-testing splits - 10% training -90% testing, 30% training -70% testing and 50% training-50% testing. The results are as follows:

- Forest types : For 10% training with cross-validation, we obtained an accuracy of 64.5% for Gaussian kernel. For linear kernel, the accuracy obtained was 82.72%. For 30% training, we obtained 77.8% and 78.9% accuracies for Gaussian kernel and linear SVM respectively. For 50% training, the values were 69.88% and 82.62% respectively.
- Iris data : For 10% training with cross-validation, we obtained an accuracy of 68.18% for Gaussian kernel. For linear kernel, the accuracy obtained was lower, 40.15%. However, for 30% training, we obtained 79.41% and 69.60% accuracies for Gaussian kernel and linear SVM respectively. For 50% training, the values were 83.33% and 79.16% respectively.
- Breast Cancer: For 10% training with cross-validation, we obtained an accuracy of 97.29% for Gaussian kernel. For linear kernel, the accuracy obtained was 97.29%. However, for 30% training, we obtained 96.92% and 97.74% accuracies for Gaussian kernel and linear SVM respectively. For 50% training, the values were 97.41% and 97.98% respectively.
- Optical Digits: For 10% training with cross-validation, we obtained an accuracy of 80.84% for Gaussian kernel. For linear kernel, the accuracy obtained was 89.17%. However, for 30% training, we obtained 93.5% and 90.72% accuracies for Gaussian kernel and linear SVM respectively. For 50% training, the values were 95.35% and 92.56% respectively.

In the implementation, we faced issues owing to the matrix $\left(\tilde{K} + C \sum_{n=1}^{N} \sum_{l \neq k} \left(\frac{y_{nk}}{2 z_{nlk}} \right) K_n K_n^T \right)$ being close to singular. This problem was solved by adding ϵI matrix to the above matrix before inverting.

3 SVM implementation using *libsvm* package

Libsvm is a library for Support Vector Machines (SVMs) that was developed in the year 2000. It is one of the most widely used SVM softwares. Libsvm modules support SVM classification for both two-class as well as multi-class. Unlike the multi-class kernel SVM implementation discussed in the previous section, libsvm implements the "one vs one" approach for multi-class classification. Therefore, if K is the total number of classes, $K(K-1)/2$ classifiers are constructed and each one is trained using the data from two classes. The objective function to be minimized for the two class problem between classes i and j is the following :

$$
\begin{aligned}
E(\mathbf{\Theta^{ij}}, \mathbf{\Theta_0^{ij}}) = & \min_{\Theta^{ij}, \Theta_0^{ij}, \xi^{ij}} \frac{1}{2}\langle \Theta^{ij}, \Theta^{ij} \rangle + C \sum_{n=1}^{N} (\xi^{ij})_n \\
\text{subject to } & (\Theta^{ij})^T \Phi(x_n) + \Theta_0^{ij} \geq 1 - \xi_n^{ij} \qquad \text{if } x_n \text{ is in } i^{th} \text{ class}, \\
& (\Theta^{ij})^T \Phi(x_n) + \Theta_0^{ij} \leq -1 + \xi_n^{ij} \qquad \text{if } x_n \text{ is in } j^{th} \text{ class}, \\
& \xi_n^{ij} \geq 0
\end{aligned}
$$

Here, $\Phi(.)$ is any chosen kernel and C is a regularization parameter, that may be optimized using cross validation.

3.1 Testing Procedure

The above objective function for binary classification in the primal form can be converted to its dual formulation by constructing the Lagrangian for a constrained optimization problem. On solving the dual problem, the optimal weight vector Θ^* satisfies the following property.

$$\Theta^* = \sum_{n=1}^{N} y_n \alpha_n \Phi(x_n)$$

where, $y_n \in \{1, 1\}$ is the class label for the pattern x_n. The decision function is the following:

$$\text{sgn}((\Theta^*)^T \Phi(x_n) + \Theta_0^*) = \text{sgn}\left(\sum_{n=1}^{N} \alpha_n y_n \Phi(x_n) + \Theta_0^* \right)$$

For classification of the test data, a voting strategy is used. Each binary classification is considered a vote and ultimately, an incoming test pattern x_t is classified as belonging to the class that received the highest number of votes.

3.2 Results and discussion

SVM classifiers using *libsvm* were constructed for classification of the four data sets (*iris,optical digits,forest type and breast cancer*)obtained from the UCI repository. All the data sets were trained and tested with the following training-testing ratios - 70% training - 30% testing and 50% training - 50% testing . The training data samples also include the data used for cross validation. Cross validation procedure is discussed in detail in the following section. Tests were performed using a linear as well as a Gaussian kernel. The following sections discuss the results obtained.

- Iris dataset - The iris data set has 3 different classes. For 50% training - 50% testing gives an accuracy of 95.55% on the test samples, where as 30% training - 70% testing gives an accuracy of 91.8%. On using a Gaussian kernel, it gives an accuracy of 95.63% and 95.34% respectively for the above split, on the test samples.
- Optical digits - It classifies all digits into 10 classes with an accuracy of 97.45% for 50% training - 50% testing split. For 30% training, it marginally reduces to 97.25% for the linear SVM case. This dataset produced much lower accuracy rates when a Gaussian kernel was used.
- Forest types - classifies the data into four classes, namely $\{d, h, s, o\}$. For 30% training, an accuracy of 86.84% was obtained. For 50%, the accuracy was 87.47%. Using a Gaussian kernel, the accuracy slightly reduces to 84.42% and 86.48% for 30% and 50% training respectively.
- Breast cancer data sets are easier to classify as it is binary. For 30% training, we obtained a higher accuracy of 96.42% and 50% training predicts the right label 96.74% of the times. Using a Gaussian kernel, the accuracy slightly reduces to 94.86% and 94.88% for 30% and 50% training respectively.

3.3 Scaling and Cross validation

In all the four datasets, the data on each attribute was scaled to lie in the range $[0, 1]$. Scaling was performed on both the training and testing data. The main advantage of scaling is to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges. Another advantage is to avoid numerical difficulties during the calculation.

Cross-validation is essential in SVM as there are a number of free parameters, that need to be fine tuned to optimum values to improve the prediction accuracy. The parameter C in equation 3 is a regularizing parameter, that ensures that the number of outliers in the training set is minimized. A Gaussian kernel has an additional parameter γ, where

$$K(x_m, x_n) = \exp\left(\gamma \|x_m - x_n\|^2\right)$$

Cross-validation procedure can prevent the overfitting problem. In our experiment, 5-fold cross-validation was performed on the training data samples. A grid-search was performed on the parameters (C, γ) and the values that give the best cross-validation accuracy is picked. Figure 1

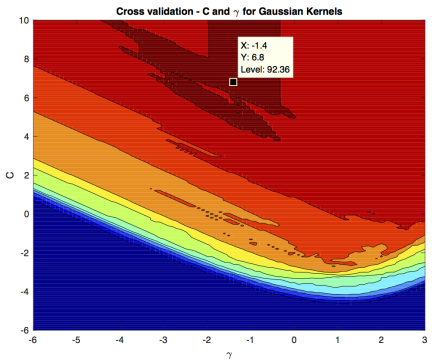


Figure 1: Contour plot with C and γ for the forest types data set showing the optimal value that gives the highest cross-validation accuracy

4 SVM with PCA Pre-Processing

4.1 Algorithm and Implementation Details

PCA is a dimensionality reduction technique used to alleviate the 'Curse of Dimensionality' which merely says that more features does not necessarily imply better classification. PCA basically transforms a set of correlated measurements into another basis where they are uncorrelated. Only the most varying features of these principal components are retained. It is observed that eigen vector corresponding to the largest eigen value is the one that portrays the highest variance and so on. PCA is one such method which essentially transforms a bunch of correlated components into a smaller number of uncorrelated principal components. The notion is that you cannot throw out data that are correlated because each part of the data reveals information about the other. Therefore, PCA initially transforms it into a set of coordinate system where the components get uncorrelated and then only the maximum varying portion of the data could be retained. This transformation onto a low dimensional subspace i.e. results in eigen space which saves tremendous computation load and leads to better classification by throwing our bad features. Throwing our features is also a way to alleviate overfitting.

Let $X \epsilon R^{NxD}$ be the feature matrix where the features are along the rows. Then PCA involves find it's covariance matrix, extracting it's eigen vectors and projecting the data onto it.

$$Cov = X^T X$$

$$U = eig(Cov)$$

Take the l largest eigen vectors where $l < D$ to get $U_{reduced}$ which forms your subspace on which the data is to be projected to obtain $X_{reduced}$

Training set accuracies	10% Training	30% Training	50% Training
Iris Dataset	99.36	97.77	97.33
Forest Dataset	98.11	97.45	93.51
Breast Cancer Dataset	97.14	97.61	97.42
Optical Digits Dataset	99.64	99.78	99.64

Test set accuracies	10% Training	30% Training	50% Training
Iris Dataset	98.51	97.14	98.66
Forest Dataset	78.27	85.60	90.34
Breast Cancer Dataset	97.15	96.74	97.34
Optical Digits Dataset	79.07	79.59	78.23

Table 1: Classification Accuracies - Training set and Testing set for PCA-SVM

$$X_{reduced} = XU$$

It is assumed that X is centered.

4.2 Results

The PCA model was implemented from scratch while SVM was run from the libSVM3.21 package. The environment used is Windows 8.1 Intel i7 processor with 8GB RAM. The models were run for 10% training - 90% testing to 50% training - 50% testing on the iris dataset, forest dataset, breast dataset and optical digits dataset. The cross validation model built onto the library was directly used here.

4.2.1 Classification Accuracies

Results are reported for 10% training - 90% testing, 30% training - 70% testing and 50% training - 50% testing. 10 random trials were run and the mean of the classification accuracies below in Table 3. Here, for PCA only 40% of the features were retained. Datasets Iris, Forest and Breast cancer show a superior performance when compared to an SVM-only model as reported in the libSVM section of the report. This implies that the features removed were 'weak' features and also variance measure is a good discriminating trait for these datasets. However for the Optical digits dataset, the results obtained by feature reduction are inferior which implies the removed features resulted in loss of discriminatory information.

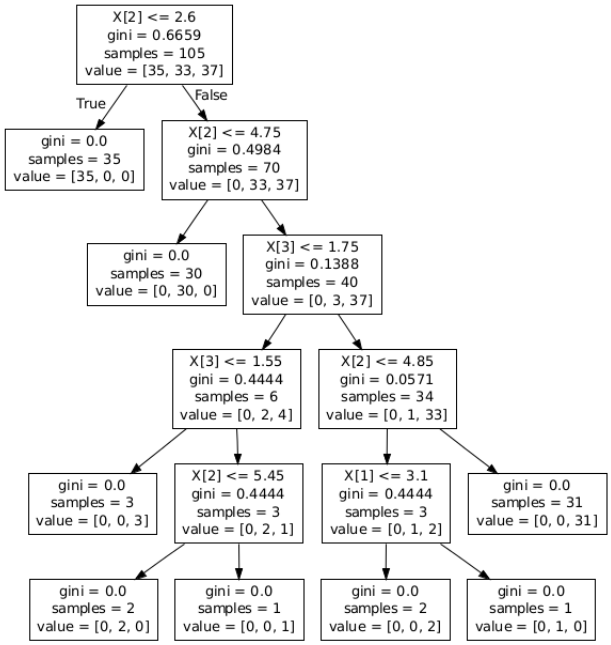
5 Decision Trees

5.1 Algorithm and Implementation Details

Decision Trees are a non-parametric method used for classification. It is easy to understand and interpret and does not require data preparation like normalization. It can handle both numerical and categorical data. However it does not generalize well. Thus in in general decision trees have low bias and high variance. The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and thus most trees learn by greedy algorithm that follow a top down approach.

In order to determine best split at a node (decision variable), gini impurity or entropy can be used. Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the subset. Gini impurity can be computed by summing the probability f_i of each item being chosen times the probability $1 - f_i$ of a mistake in categorizing that item. It reaches its minimum (zero) when all cases in the node fall into a single target category. Similarly entropy can be defined as $I_E(f) = -\sum_{i=1}^{m} f_i \log_2 f_i$. Information gain is thus Entropy(Parent) - Weighted sum of Entropy (Children).

Sample tree (based on gini impurity) on iris dataset is shown below



5.2 Performance

70 % of the dataset was used for training and 30 % for testing. The criteria for split whether entropy gain or gini impurity were cross validated. They are other variables which can be cross validated like pruning criteria or depth of tree but we did not consider these. Entropy was consistently found as a better criteria for split across all datasets. The data results show that trees are not the best method for classification

Forest types: Test set accuracy was 86 %.
Breast cancer: Test set accuracy was 90 %.
Optical digits: Test set accuracy was 90 %.
Iris: Test set accuarcy was 95 %.

6 Random Forests

6.1 Algorithm and Implementation Details

Random Forest is an ensemble method learning for classification and regression. In this report we are only considering it for classification. The method combines bagging (model averaging) and random selection of features to construct a collection of deccison trees with controlled variance. The algorithm can be formulated as

```
1. For b = 1 to B:
       (a) Draw a bootstrap sample Z of size N from the training data.
       (b) Grow a random-forest tree Tb to the bootstrapped data, by recursively
           repeating the following steps for each terminal node of the tree, until the
           minimum node size is reached
                   i. Select m variables at random from the p variables.
                   ii. Pick the best variable/split-point among the m
                   iii. Split the node into two daughter nodes.
2. Output the ensemble of B trees {T_b}
```

To make a prediction at a new dataset x, let Cb be the class prediction of bth random-forest tree. Take majority vote for from all the trees in the forest.

The default value of m i.e number of features in each tree is sqrt(p), where p is the features in our dataset. Since trees are noisy, they benefit from the averaging. Moreover, since each tree generated in bagging is identically distributed (i.d.), the expectation of an average of B such trees is the same as the expectation of any one of them. This

means the bias of bagged trees is the same as that of the individual trees, improvement is obtained through variance reduction. Although the number of trees can be seen as hyperparameter and cross validation used to find number of trees. In actual practice it is not needed since trees are sampled identically. Literature shows the performance is best for 100-200 trees.

6.2 Performance

70 % of the dataset was used for training and 30 % for testing. Random forest are invariant to monotonic transformation of individual features and thus no preprocessing step like normalization or scaling is needed. Number of trees was used as cross validation parameter and number of trees were increased by 50 i.e 1,51,101,...,301. All datasets gave best performace on the training set with 101,151 or 201 trees. Error was calculated on the test set and also as k (5) fold cross validation on the traing set with the with best estimate for number of trees between 101 to 251.

Forest Types: Test set accuracy was 90 % and it was 91 % on k fold cross validation. Showing that model generalizes well.
Breast cancer: Test set accuracy was 97 % and 97 % as well on k fold cross validation.
Optical digits: Test set accuracy was 98 % and 98 % as well on k fold cross validation.
Iris: Test set accuarcy was 95 % and number of trees was 51

7 Gradient Boosting Trees

7.1 Algorithm and Implementation Details

Gradient boosting is a technique that produces a prediction model in the form of an ensemble of weak prediction models. It is called gradient boosting because of its interpretation as a optimization algorithm (iterative function gradient descent) on a suitable cost function. That is, algorithms that optimize a cost function over function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction.

Thus we fit an additive model (ensemble). If we consider Adaboost as $H(x) = \sum_t p_t h_t(x)$, then it can be seen as addition of gradient descent and boosting. In Gradient Boosting we

- Fit an additive model $\sum_t p_t h_t(x)$ in a forward stage-wise manner.
- In each stage, introduce a weak learner to compensate the shortcomings of existing weak learners.
- In Gradient Boosting, shortcomings are identified by gradients, whereas in Adaboost shortcomings are identified by high-weight data points. But both tell us how to improve our model in each successive iteration.

The depth of each estimator is the most important parameter used to tune the model. The number of boosting stages to perform is also a parameter which needs to be adjusted and it controls the maximum allowed level of interaction between variables in the model. Regularization is important to ensure the model generalizes well. The depth of the estaimator number of gradient boosting iterations M (i.e. the number of trees in the model when the base learner is a decision tree) can be cross validated. Increasing M reduces the error on training set, but setting it too high may lead to overfitting. Other techinques like shrinkage and penalizing complex trees can also be used for regularization.

7.2 Performance

70 % of the dataset was used for training and 30 % for testing. Boosting trees also generally dont need normalization. Number of estimators (boosting stages to perform) and depth of individual estimators were used as cross validation parameter. k (5) fold cross validation was used. Number of estimators was increased by 50 i.e 1,51,101,...301 and depth was changed from 1, 3, ...,10. The performance was best when the number of estimators varied from 101 to 251. But the depth remained constant at 3. 3 is generally considered the best default depth for gradient boosting trees.

Forest types: Test set accuracy was 89 % and it was 90 % on k fold cross validation. Thus it proves that gradient boosting trees dont generally overfit the data.
Breast Cancer: Test set accuracy was 95 % and 96 % on k fold cross validation.
Optical digits: Test set accuracy was 97 % and 97 % on k fold cross validation.
Iris: Test set accuracy was 97 % and number of estimators and depth both were 1.

8 Logistic Regression

8.1 Algorithm and Implementation Details

Logistic Regression is a classification algorithm inspite of it's name, based on the logistic function used on the hypothesis. The difference between regression and classification lies in the continuous nature and the discrete nature of the labels respectively. Since classification is not a linear function, the role of linear regression is limited.

The hypothesis $h_\theta(x)$ needs to be in the range $0 < h_\theta(x) < 1$. This is realized by using a sigmoid function $g(z)$ as shown in Figure 1, which translates every real value into the range $0 to 1$.

Therefore we have,

$$h_\theta(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

where $z = \theta^T x$

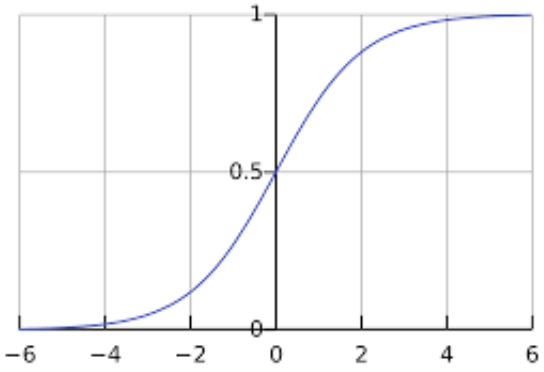


Figure 2: Sigmoid Function

[H]

The cost function is framed intuitively by imposing a large penalty on wrong decisions. It is given by (for one sample) as shown below:

$costFn = -ylog(h_\theta(x)) - (1-y)log(1-h_\theta(x))$.

It can be seen that when $y = 1$ and $h_\theta(x) = 0$, $costFn$ goes to infinity. Similarly, when $y = 0$ and $h_\theta(x) = 1$, $costFn$ $J(\theta)$ goes to infinity. Generalizing, the entire cost function is given by (10) and it's vectorized version is given by (**??**).

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}log(h_\theta(x^{(i)})) - (1-y^{(i)})log(1-h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \tag{9}$$

$$J(\theta) = -\frac{1}{m}[log(g(X\theta))^T y - log(1-g(X\theta))^T(1-y)] + \frac{\lambda}{2m} < \theta, \theta > \tag{10}$$

Note that here,

X is the feature matrix

y is the class labels

θ is the machine

n is the dimension of the machine

m is the number of samples used for training

$\lambda$is the regularization parameter

$< ., . >$denotes inner product

The optimum value for the machine is obtained by running gradient descent on the above cost functions. Note that the regularization term does not have the $\theta_0$term and gradient descent equation is turn separately for the $\theta_0$term. These are shown in (11) and (12) and are run repeatedly until convergence.

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)} - y^{(i)}) x_0^i \tag{11}$$

$$\theta_0 := \theta_0 - \alpha [(\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)} - y^{(i)}) x_j^i) + \frac{\lambda}{2m} \theta_j \tag{12}$$

where, $j \epsilon \{1, 2, ...n\}$.

8.1.1 Regularization

In (10) and (**??**), the second term is a regularizer term which eliminates overfitting of the data by making sure θ is not too large (norm). Now, to reduce the value of the cost function to zero, θ needs to be very small which would reduce the regularizer term in the objective function. λ is known as the regularization parameter which signifies how much the costs of the θ parameter are inflated. One has to be careful in making sure λ is not too large which may smooth out the function to a large degree and may result in underfiting. As λ increases, the optimum value of θ reduces.

8.2 Results:

The models were implemented from scratch in Matlab. The environment used is Windows 8.1 Intel i7 processor with 8GB RAM. The models were run for 10% training - 90% testing to 50% training - 50% testing on the iris dataset, forest dataset, breast dataset and optical digits dataset.

8.2.1 Choice of lambda λ

The free parameter λ was selected by running a k-Fold cross validation on the validation set which is a subset of the original training set. Here, the training set was divided into a validation - testing set and a validation - training set. k was chosen to be 15. Therefore, 15% of the training data was used for validation testing and remaining for validation training. This was run for 10 iterations where a different subset of k samples were chosen to be validation testing and the remaining for validation training. The mean of the best λ generated from all the iterations were picked to build the final model on which the test set is tested. λ values were chosen from the range 0.001 to 2. The results for this is shown in Figures 2-5 for the various datasets.

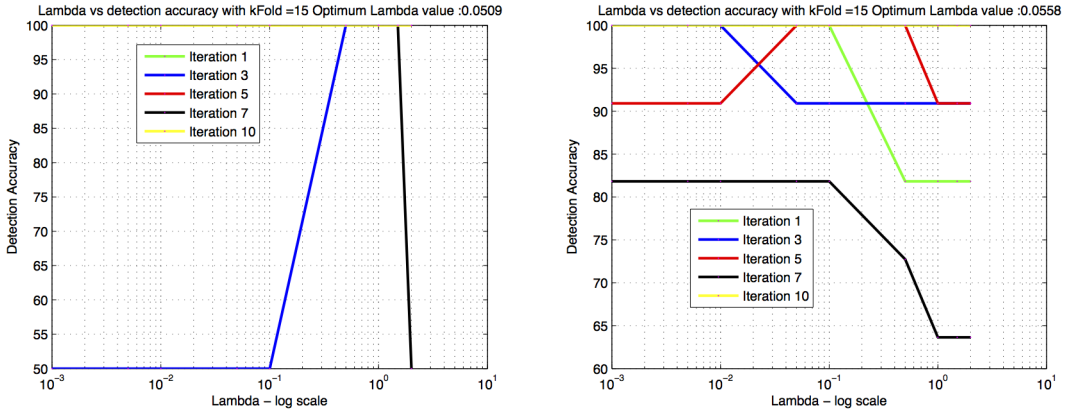


Figure 3: Justification for the choice$\lambda$for 10%training and 50% training (L-R) - Iris Dataset

[H]

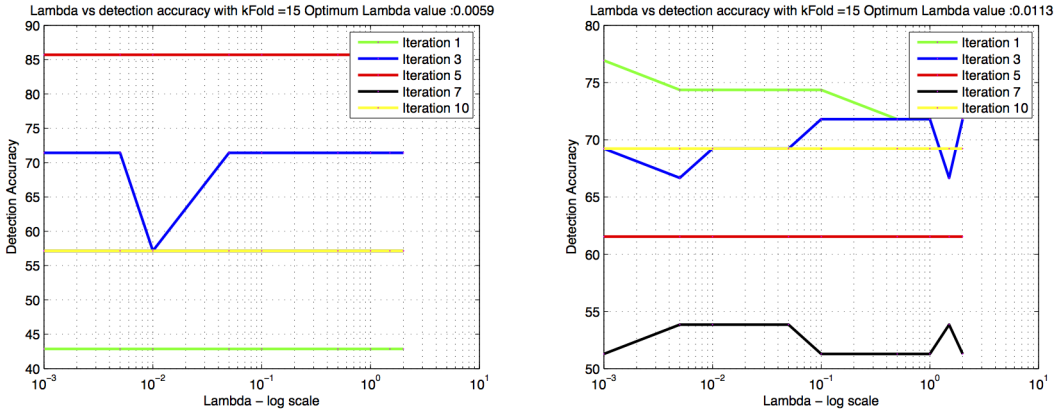


Figure 4: Justification for the choice$\lambda$for 10%training and 50% training (L-R) - Forest Dataset

[H]

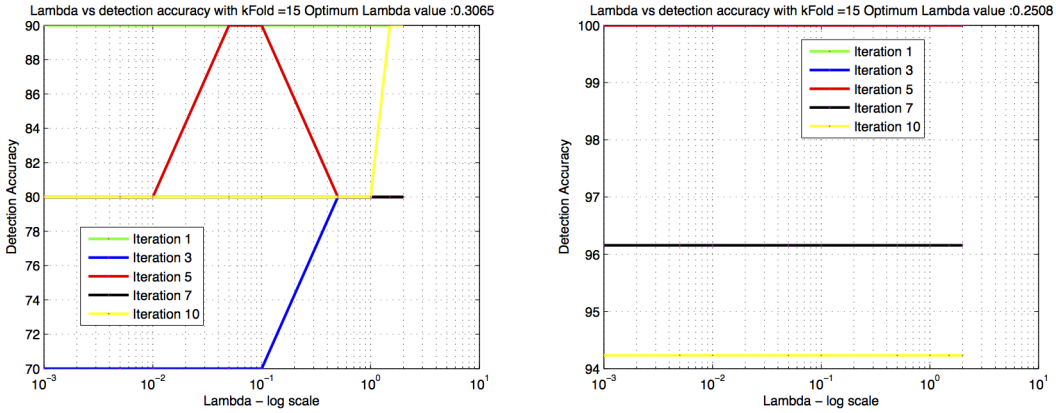


Figure 5: Justification for the choice$\lambda$for 10%training and 50% training (L-R) - Breast Dataset

[H]

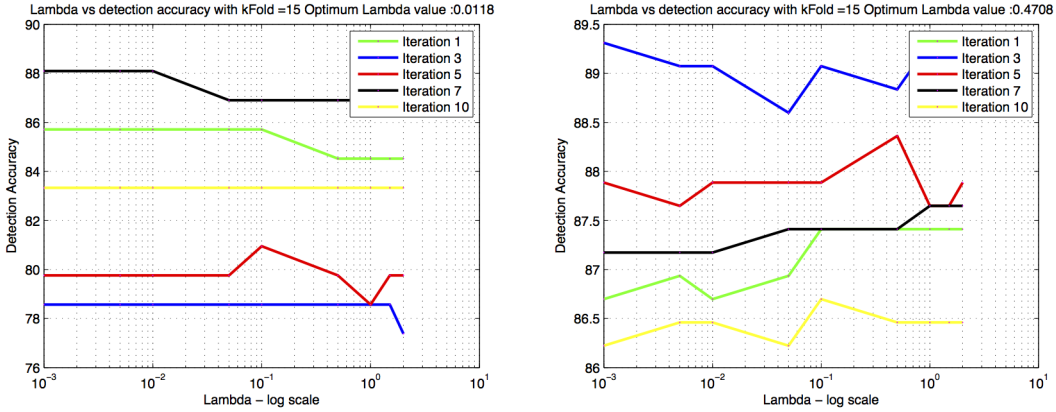


Figure 6: Justification for the choice$\lambda$for 10%training and 50% training (L-R) - Optical Dataset

[H]

Training set accuracies	10% Training	30% Training	50% Training
Iris Dataset	93.33	97.78	98.67
Forest Dataset	62.27	66.87	70.08
Breast Cancer Dataset	98.57	98.57	97.43
Optical Digits Dataset	90.21	89.44	89.00

Test set accuracies	10% Training	30% Training	50% Training
Iris Dataset	90.37	92.38	95
Forest Dataset	58.28	62.42	64.04
Breast Cancer Dataset	95.56	96.34	96.69
Optical Digits Dataset	83.17	85.22	86.26

Table 2: Classification Accuracies - Training set and Testing set for Logistic Regression

Computation Time in seconds	10% Training	30% Training	50% Training
Iris Dataset	32.71	34.10	35.76
Forest Dataset	45.93	49.37	122.17
Breast Cancer Dataset	24.54	26.29	27.06
Optical Digits Dataset	365.60	578.39	1075.42

Table 3: Computation Time - Logistic Regression

8.2.2 Classification Accuracies

Results are reported for 10% training - 90% testing, 30% training - 70% testing and 50% training - 50% testing. 10 random trials were run and the mean of the classification accuracies below in Table 1.

8.2.3 Computation Times

Table 2 shows the time taken for Logistic Regression.

9 Deep Learning using Multilayer Perceptron

In our implementation of the deep learning algorithm, we make use of a multilayer perceptron model with one hidden layer. The number of nodes in the input layer of the model will be the same as the number of features or attributes in the data sample and the number of output nodes will be equal to the number of class labels. Figure 7 illustrates a schematic diagram of a multilayer perceptron with one hidden layer. The number of nodes in the hidden layer was arbitrarily chosen as 25.

The cost function to be minimized with the regularization term is given as follows:

$$J(\Theta) = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\Big[-y_n^{(k)}\log(h_\Theta^k(x_n)) - (1-y_n^{(k)})\log(1-(h_\Theta^k(x_n)))\Big] + \frac{\lambda}{2m}\left[\sum_{j=1}^{hl}\sum_{k=1}^{d}(\Theta_{j,k}^{(1)})^2 + \sum_{j=1}^{K}\sum_{k=1}^{hl}(\Theta_{j,k}^{(2)})^2\right] \quad (13)$$

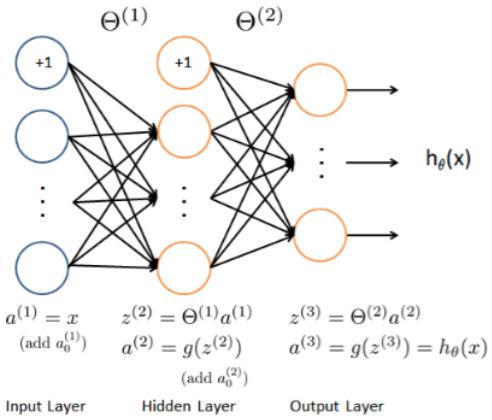


Figure 7: Schematic diagram of a multilayer perceptron model with 1 hidden layer

Backpropagation algorithm is employed for training the weight matrices $\Theta^{(1)}$ and $\Theta^{(2)}$ corresponding to the input layer and hidden layer respectively. The intuition behind the backpropagation algorithm is the following. Given a training example (x_n, y_n), we will first run "forward pass" to compute all the activations throughout the network, including the output value of the hypothesis $h_\Theta(x)$. Then, for each node j in layer l, we would like to compute an error term $\delta(l)$ that measures how much that node was "responsible" for any errors in the output. Using chain rule, a gradient descent algorithm can be derived for updating the weight vectors. In the acutal MATLAB implementation, a built-in function *fmincg* was used for optimization.

9.1 Testing Procedure

For testing, the test data is assigned the class label corresponding to the index of the output node k, that maximizes the hypothesis function with the trained weight vectors.

$$C_k(x_t) = \arg\max_k h_\Theta^k(x_t)$$

9.2 Results and discussion

- Iris dataset - The iris data set has 3 different classes. For 50% training - 50% testing gives an average accuracy of 96% on the test samples, where as 30% training - 70% testing gives an average accuracy of 94% over 10 trials. Training on 10% of the data produces an accuracy of only about 65%.
- Optical digits - It classifies all digits into 10 classes with an accuracy of 94.68% for 50% training - 50% testing split. For 30% training, it marginally reduces to 94.37% . Even for 10% training, the classifier gives an accuracy of 95.13%.
- Forest types - classifies the data into four classes, namely $\{d, h, s, o\}$. For 30% training, an accuracy of 90.16% was obtained. For 50%, the average accuracy was 90.38%. For 10% training and remaining 90% testing, an average accuracy was about 85%.
- Breast cancer data sets are easier to classify as it is binary. For 30% training, we obtained a higher accuracy of 97.34% and 50% training predicts the right label 95.98% of the times. 10% training produced 96% accuracy.

9.3 Cross-validation with regularizing parameter λ

The parameter λ is the regularizing parameter that ensures that the weight vectors do not take huge values. 5 fold cross-validation was done on the training data samples to obtain the optimum value of λ that gave the best cross-validation accuracy. Figure 8 plots the cross-validation accuracy obtained for various λ values for 30% and 50% testing. It may be seen that for higher values of λ, cross-validation accuracy falls steeply.

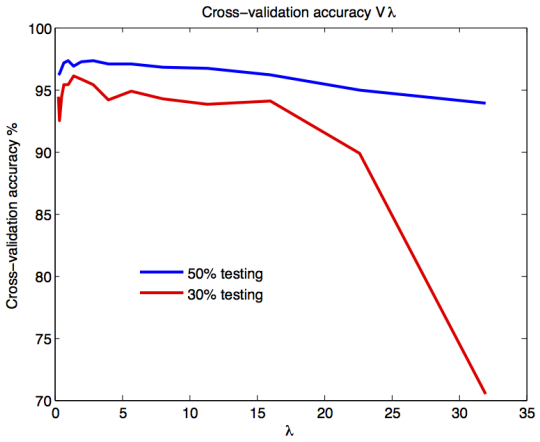


Figure 8: Cross-validation accuracy for various λ values

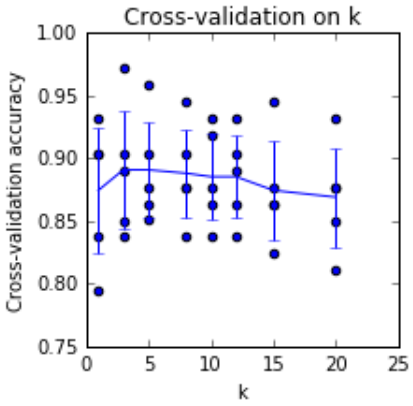
10 k-Nearest Neigbors

10.1 Algorithm and Implementation Details

k Nearest neighbour is a simple non parametric classfication algorithm that per se does not involve learning in the traditional sense. We have a set of training data and when a test data comes, we compute distance between test data and all training data. Based on the distance metric and majority vote of the k closest training data labels,test data is assigned its label. The k value itself is ahyper parameter which can be calculated using cross validation.

10.2 Performance

70 % of the dataset was used for training and 30 % for testing. Out of the 70 % training data 5 fold cross validation was performed on the dataset to find the best value of k. A sample plot of cross validation is shown for forest types data. Similar procedure was done for other datasets to find the best value of k.



Forest types: Test set accuracy was 87 % and k was 3.
Breast cancer: Test set accuracy was 98 % and k was 5.
Optical digits: Test set accuracy was 98 % and k was 5.
Iris: Test set accuracy was 98 % and k was 3

11 Naive Bayes

11.1 Algorithm and Implementation Details

Naive bayes is a probabilistic classifier based on applying bayes theorem with independence assumptions between the features.If our data has n features, $x_1, x_2, x_3...x_n$, and Kclasses, it assigns $p(C_k|x_1, ..., x_n)$ for each of the K possible outcomes. Using bayes theorem, the conditional probability can be written as

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

i.e is

$$posterior = \frac{prior * likelihood}{evidence}$$

Using conditional independence, the the probability can be written as

$$p(C_k|x_1, x_2, ...x_n) = \frac{1}{z}p(C_k)\prod_{i=1}^{n} p(x_i|C_k)$$

where z is just a scaling factor. Thus the class label is assigned as

$$\arg\max_{k\in 1...K} p(C_k)\prod_{i=1}^{n} p(x_i|C_k)$$

Here we have implemented multinomial naive bayes which is a specific instance of naive bayes classifier that uses a multinomial distribution (generalization of binomial distribution) for each of the features.

11.2 Performance

70 % of the dataset was used for traing and 30 % for testing.

Forest types: Test set accuracy was 83 %.
Breast cancer: Test set accuracy was 91 %.
Optical digits: Test set accuracy was 90 %.
Iris: Test set accuracy was 93 %.

12 Conclusion

13 Conclusion

The following table summarizes the best performance observed using the different algorithms on the four datasets:

Algorithm	Forest Types	Breast Cancer	Optical Digits	Iris
Multiclass Kernel SVM	82.72%	97.98%	95.35%	83.33%
LibSVM Package	87.4 %	96.7%	97.4%	95.6%
Decision Trees	86 %	90 %	90 %	95 %
Random Forest	90 %	97 %	98 %	95 %
Gradient Boosting Trees	89 %	95 %	97 %	97 %
Deep Learning	90.3%	97.3%	94.6%	96%
Multinomial Logistic Regression	64.04%	96.69%	86.26%	95%
K nearest neighbours	87 %	98 %	98 %	98 %
Naive Bayes	83%	91%	90%	93%