Machine Learning Course Project

Jennifer Cheung, Joshua Kirstein,
Abhishek Kumar, Abhishek Mohanty, Abhinav Rathi
University of Florida
{*jennifer.cheung, joshkirstein, abhishekakumar, avisec, rathi*}*@ufl.edu*

***Abstract*—In this paper we explore a handful of machine learning classification algorithms and compare their performance. A multi-class kernel SVM was built from scratch and compared to scikit-learn's SVM. We also compare the classification algorithms together and evaluated their trade-offs.**

***Index Terms*—neural networks, svm, kNN, kmeans, random forests, classification, machine learning**

I. Installation

To run the code samples in this project, python and some extra libraries are required. To install the libraries run the following commands:

```
sudo pip install pandas
sudo pip install keras
sudo pip install cvxopt
sudo easy_install --upgrade six
```

II. Usage

Simply call

```
python main.py
```

or

```
python kernel_svm_test.py
```

NOTE: A sequence of plots will be shown. In order to progress through this sequence, you must close the plot that's currently being displayed.

III. Multi-Class Kernel SVM

A multi-class kernel support-vector machine was implemented. We build a multi-class kernel SVM as K two-class kernel SVMs, using the one-against-all schema to classify an incoming pattern x (that is, choose the class k who's two-class kernel prediction gives the highest value). A two-class kernel SVM trained with N patterns is constructed by solving the following convex quadratic programming optimization task (using CVXOPT):

$$\begin{aligned} \underset{\{\lambda_n\}}{\text{maximize}} \quad & \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \kappa(x_i, x_j) \\ \text{subject to} \quad & 0 \le \lambda_i \le C,\ i = 1, \ldots, N. \\ & \sum_{i=1}^{N} \lambda_i y_i = 0 \end{aligned}$$

where C is a constant that helps with outlier rejection and $\kappa(x, y)$ is the kernel function. Once we have $\{\lambda_n\}$ we compute θ_0 as the average of the following values for each support vector i :

$$\theta_{0,i} = \frac{1}{y_i} - \sum_{j=1}^{N} \lambda_j y_j \kappa(x_j, x_i)$$

Finally, to classify an incoming pattern x we simply compute

$$y(x) = \sum_{i=1}^{N} \lambda_i y_i \kappa(x_i, x) + \theta_0$$

A. Limitations

There are several limitations to this implementation. First of all, the kernel implementation of an SVM is inherently slower due to the kernel computations. Specifically in our implementation, we must compute the Gram matrix for the kernel over the samples — a computation which is quadratic in the number of patterns. Classifying patterns is similarly deficient; looking at the formula for prediction above, we must compute N kernel values. Varying the kernel changes the performance of the algorithm. Our choice of using python (an interpreted language) also slows down these computations greatly.

Accuracy does not seem to be hindered from this implementation. We use an industrial strength convex optimizer (CVXOPT), which gives much better results than the suggested majorized algorithm. Compared to *libsvm*, our implementation of a multi-class kernel SVM is more accurate (albeit *significantly* slower). Using a Gaussian kernel provided the most accurate classification whilst being the slowest. Using a Linear kernel provided mildly accurate classification whilst being the fastest.

B. Testing

We ran the multi-class kernel SVM implementation on multiple data sets. On the handwritten digit data set, our implementation takes a very long time to run (for train percentage less than 0.7 it runs in a moderate amount of time). To see the results from the other data sets, run the code.

To show that our implementation was indeed kernalized, we ran a visual experiment using two-class Gaussian data. The blue and green points are the data colored by their classes. The points in red describe points that are very close to the splitting surface generated by the two-class kernel SVM (using a Gaussian kernel with $\sigma = 20$), thus effectively plotting the splitting surface.

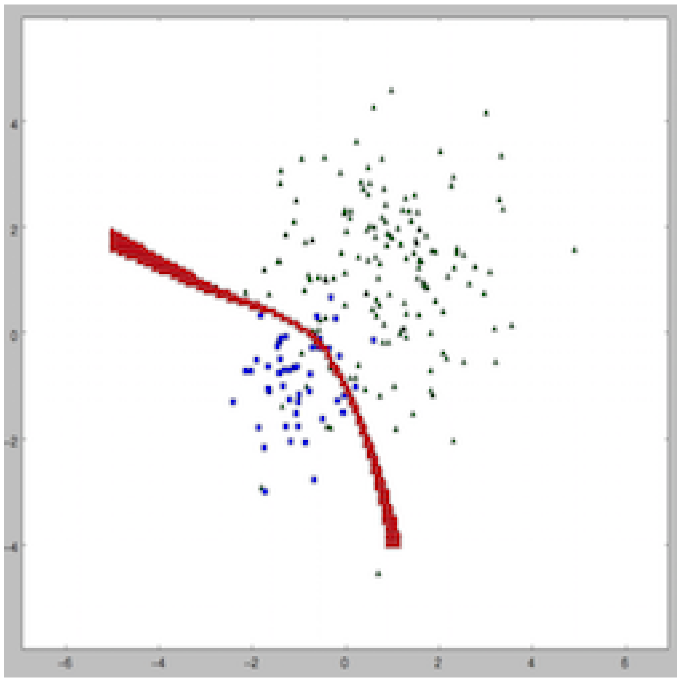


Fig. 1. Kernel visualization #1

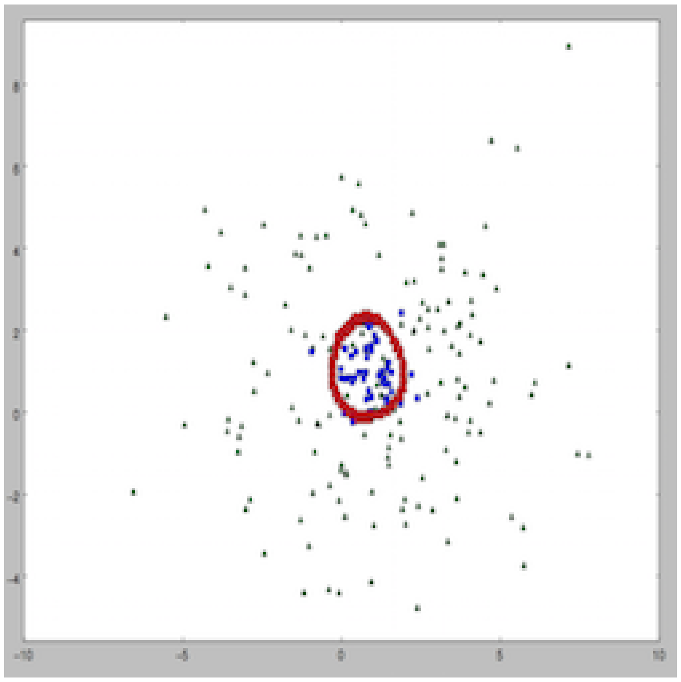


Fig. 2. Kernel visualization #2

IV. Algorithms Demonstrated

A. Neural Network

1) Introduction: Deep learning is a branch of machine learning that attempts to model data in multiple processing layers. In its most general form it's an artificial neural network architecture with densely connected layers.

2) Implementation: In this project we are using the Sequential implementation of Keras library, which contains linear stack of layers that encapsulates RNN model for deep learning. The method fit in Keras provides parameters like epoch, validation split, and callbacks which help to modify the LSTM system. Callbacks are methods which provides a view on the internal state and statistics of the model during training stage [13]. Validation split helps to adjust the fraction of data to be held-out as validation data. Epoch is the number of cycles the training data has to be moved through the system, as part of system learning. The input data is fed into the system in batches. The predict method predicts the output depending on the activation parameter, optimizer and batch-size parameter. The architecture of the model is the simplest one hidden layer – one input layer, one hidden layer and an output layer. The number of neurons in the hidden layer varies according to the dataset.

a) Preprocessing: Breast Cancer dataset: The breast cancer dataset is binary classification. The classes are represented as 2 and 4. So the classes are first converted into 0 and 1.

Digit dataset: The digit dataset is multi class with classes 0 – 9. The classes are converted into an array using keras utils to_categorical() method. This method converts class vector (integers from 0 to nb_classes) to binary class matrix with 1 for particular class and 0 for rest.

Forrest dataset: Forrest dataset is also multi class with classes d, h, s, o. The character classes are first converted to numeric classes with {'d': 0, 'h': 1, 's': 2, 'o': 3}. The numeric classes are then converted into an array using the same processing as digit dataset.

b) Details of Learning: Breast Cancer dataset: The model is trained using rmspropagation optimizer with a batch size of 32, sigmoid activation of dense layer and binary cross entropy (log loss) as objective function. There are 64 neurons in the hidden layer. The model is trained for 20 epochs.

Digit dataset: The model is trained using rmspropagation optimizer with a batch size of 32, sigmoid activation of dense layer and categorical cross entropy (log loss) as objective function as this is a multiclass classification. There are 512 neurons in the hidden layer. The model is trained for 20 epochs.

Forest dataset: The model is trained using rmspropagation optimizer with a batch size of 32, sigmoid activation of dense layer and categorical cross entropy (log loss) as objective function as this is a multiclass classification. There are 512 neurons in the hidden layer. The model is trained for 20 epochs.

B. Random Forests

1) Introduction: A random forest is a meta estimator that fits a number of decision tree classifiers on various subsamples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.

2) Implementation: For Implementation purposes here we have used scikit-learn's RandomForestClassifier module in python. The various parameters available in the module were: i) n_estimators: n_estimators represent the number of trees in the forest. In general, the more the number of trees, the better the results. However, after a certain point, the improvement decreases as the number of trees increases. This is because the benefit in prediction performance from learning more trees will be lower than the cost in computation time for learning these additional trees. Here we have tested estimator values ranging from 10 to 1000. ii) max_depth: This parameter represents the maximum depth of the tree. If none max_depth value is provided, the nodes are expanded until all leaves are pure. We are testing max_depth values ranging from 5 to 200. iii) max_features: This parameter represents the number of features to consider when looking for the best split. max_features is not included as a free parameter as we noticed that the default auto option gives the best accuracy. iv) Others parameters are set to default.

The plots below show the accuracy vs the estimator and the max_depth values, for a 0.5 test train split, on the Breast Cancer Wisconsin (Original) data set from the UCI Machine Learning repository.

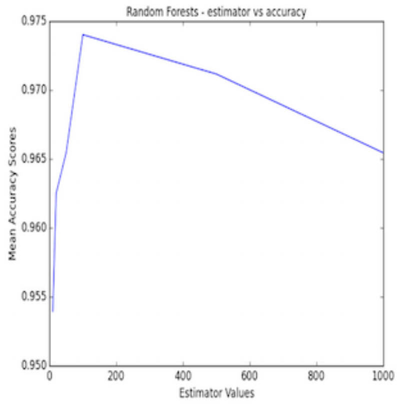


Fig. 3. Accuracy vs estimators

3) Testing: Data from all the datasets under consideration in divided into training and testing, ranging from 10% training / 90% testing to 50% training / 50% testing. The estimator and max_depth values are chosen based on the best accuracy score. We fit the Random Forests classifier on the training data and then try it on test data. The performance metric used is normalized accuracy_score. In this metric the set of labels produced must exactly match the corresponding set of label in its input. The associated plot is displayed.

The accuracy vs. above training/test data percentages on the same dataset is plotted above.

C. Support Vector Machine

1) Introduction: Support Vector Machine, or henceforth SVM, is a supervised learning algorithm that analyzes data for classification. A SVM constructs a hyperplane or a set of hyperplanes in the D-dimesnional space for this classification.

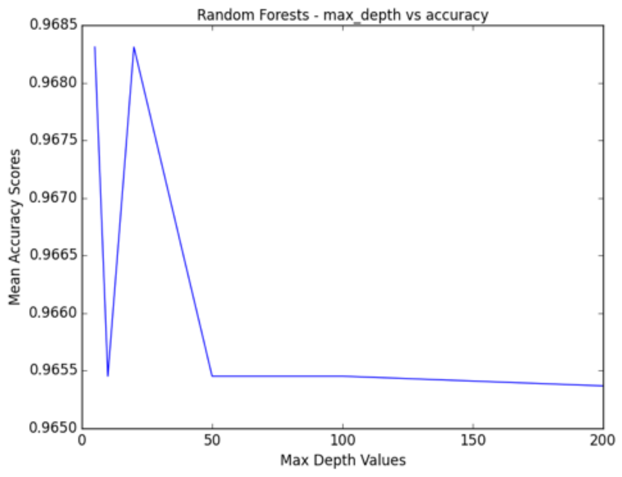


Fig. 4. Accuracy vs max_depths

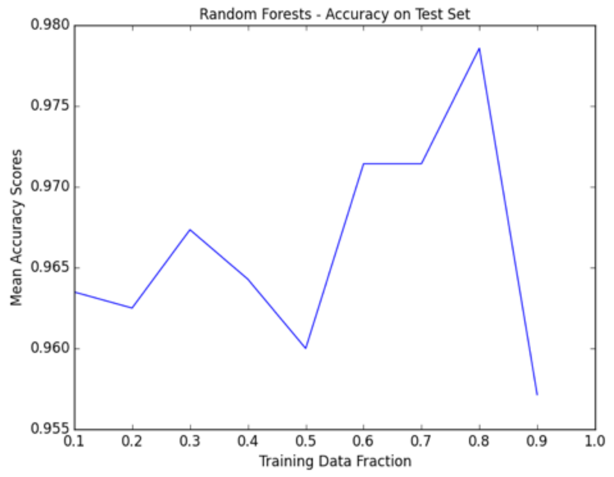


Fig. 5. Accuracy on Test Set

2) Implementation: For Implementation purposes here we have used scikit-learn's svm module in python. The implementation is based on libsvm. The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to dataset with more than a couple of 10000 samples. The multiclass support is handled according to a one-vs-one scheme.The various parameters available in the module were: i) C values: C is essentially a regularisation parameter, which controls the trade-off between achieving a low error on the training data and minimising the norm of the weights. Here we have tested a multitude of C values from 0.01 to 1000. ii) Kernel: Trying out multiple options, 'rbf' was chosen as it gives the best accuracy on test data. ('rbf' here stands for radial base function) iii) Others parameters are set to default.

We plot below the accuracy we get according to the C values (as above) on Breast Cancer Wisconsin (Original) data set from the UCI Machine Learning repository.

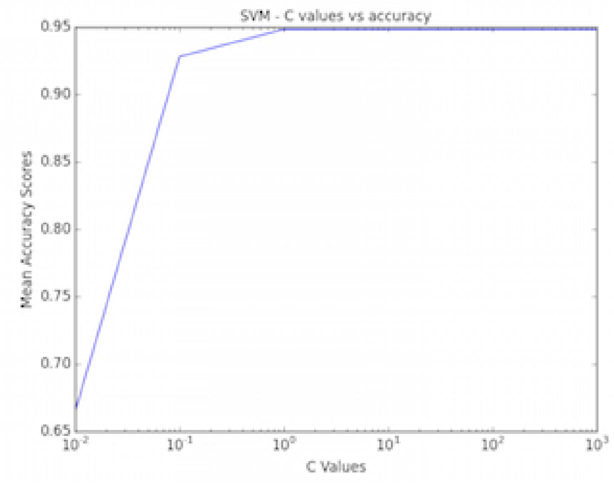


Fig. 6. Accuracy vs C_values

3) Testing: Data from all our datasets under consideration in divided into training and testing, ranging from 10% training / 90% testing to 50% training / 50% testing. C Value is chosen to giving the best accuracy score. We fit the SVM model on the training data and then try it on test data. The performance metric used is normalized accuracy_score. In this metric the set of labels produced must exactly match the corresponding set of label in its input. The associated plot is displayed.

The accuracy vs. above training/test data percentages on the same dataset is plotted.

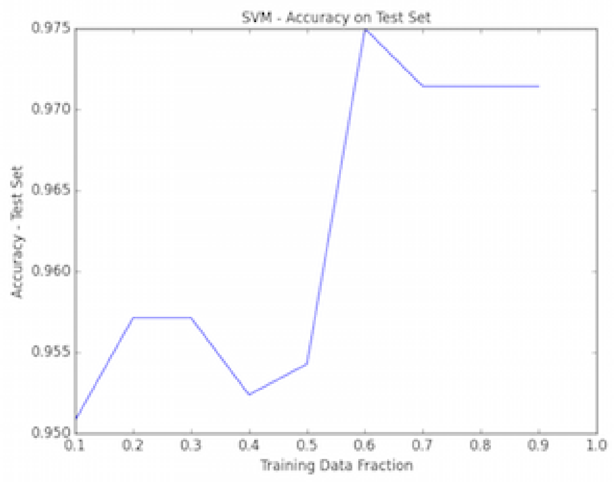


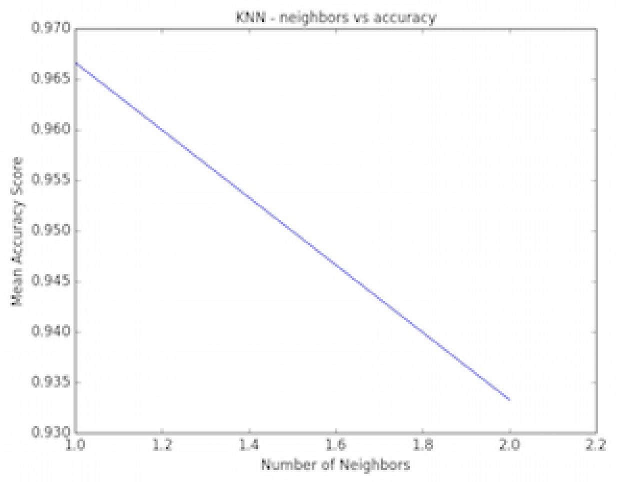
Fig. 7. Accuracy on Test State

D. K Nearest Neighbors

1) Introduction: K Nearest Neighbors is a distance based classifier that labels incoming feature vectors based on the labels of the k nearest vectors provided in the training set.

2) Implementation: The number of neighbors is cross validated to obtain the accuracies of using different numbers of neighbors. From this, the optimal number of neighbors is chosen, k. The incoming patterns are then classified by the majority of the class labels of the closest k feature vectors from the training set.

The accuracy versus the number of neighbors for the 30% training/50% test partition



3) Testing: Data from all our datasets under consideration in divided into training and testing, ranging from 10% training / 90% testing to 50% training / 50% testing. After fitting the model onto the training set, we compare the prediced class labels from the test set to the actual class labels.
The Accuracy Scores:

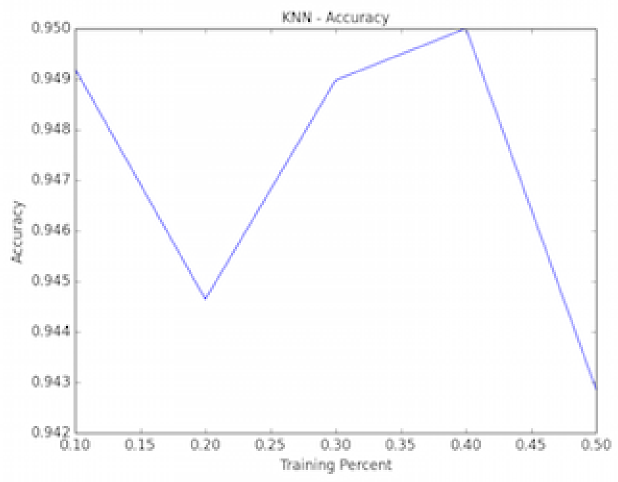


Fig. 8. KNN Accuracy

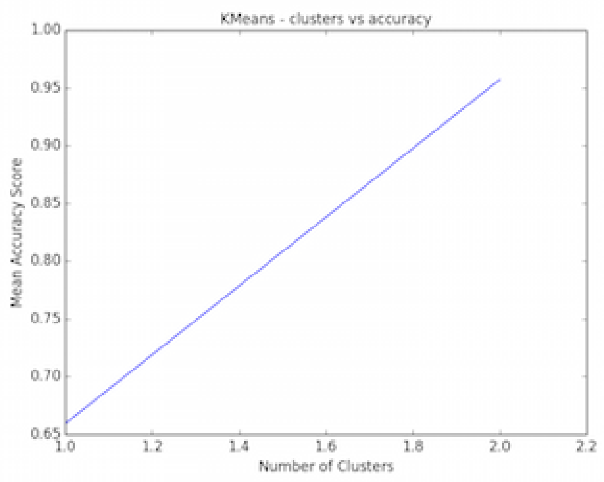
E. KMeans

1) Introduction: KMeans is a cluster based analysis that can be loosely related to KNN by using the comparison of the incoming feature vectors to the cluster centroids. KMeans calculates centroid locations, which computationally, is NP-hard, and classifies incoming feature vector on the KNN comparison, where the number of neighbors is one.

2) *Implementation:* The number of clusters is crossvalidated to obtain the accuracies of varying amounts of clusters, from one to the number of classes. From this, the optimal number of clusters is chosen, k, and the centroids, μ_k of these clusters are saved. More explicitly, μ_0 is obtained by initializing the clusters, in this case, using sklearn's kmeans++; μ_{k+1} is obtained by iterateratively taking the means of the feature vectors in the updated cluster obtained from μ_k. This continues until either $\|\mu_{k+1} - \mu_k\| < 0.0001$ or the number of iterations reaches 300, whichever comes first.

The class labels of the incoming patterns are determined by the closest centroid. That is, the class label of the closest centroid to the incoming pattern is assigned as that feature vector's class.

The accuracy versus the number of neighbors for the 30% training/50% test partition



3) *Testing:* Data from all our datasets under consideration in divided into training and testing, ranging from 10% training / 90% testing to 50% training / 50% testing. After fitting the model onto the training set, we compare the prediced class labels from the test set to the actual class labels.
The Accuracy Score:

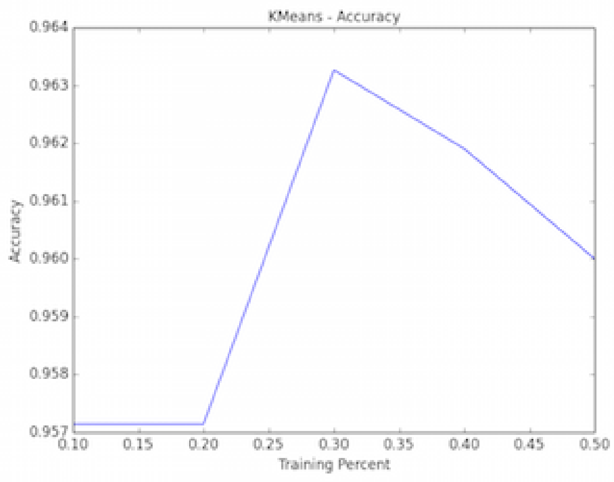
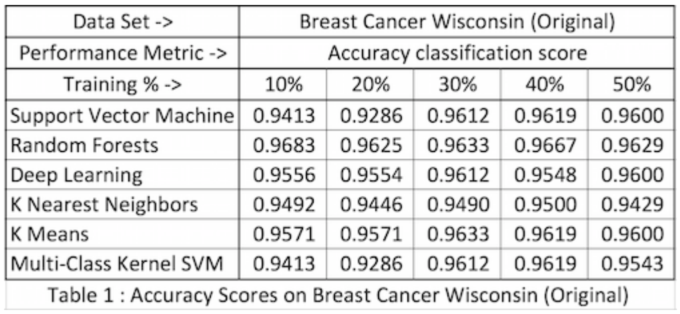


Fig. 9. Kmeans accuracy

V. Data sets

A. *Breast Cancer Wisconsin (Original)*

We are using the breast-cancer-wisconsin.data file as our training data. The following are the features present in this dataset, id, thickness, cell_size, cell_shape, adhesion, single_cell_size, nuclei, chromatin, nucleoli, mitoses, class. Since this data set has some missing values in the nuclei column, we replace those missing values by the mean of the column.



Data Set ->	Breast Cancer Wisconsin (Original)				
Performance Metric ->	Accuracy classification score				
Training % ->	10%	20%	30%	40%	50%
Support Vector Machine	0.9413	0.9286	0.9612	0.9619	0.9600
Random Forests	0.9683	0.9625	0.9633	0.9667	0.9629
Deep Learning	0.9556	0.9554	0.9612	0.9548	0.9600
K Nearest Neighbors	0.9492	0.9446	0.9490	0.9500	0.9429
K Means	0.9571	0.9571	0.9633	0.9619	0.9600
Multi-Class Kernel SVM	0.9413	0.9286	0.9612	0.9619	0.9543

Table 1 : Accuracy Scores on Breast Cancer Wisconsin (Original)

Fig. 10. Table 1 : Accuracy Scores on Breast Cancer Wisconsin (Original)

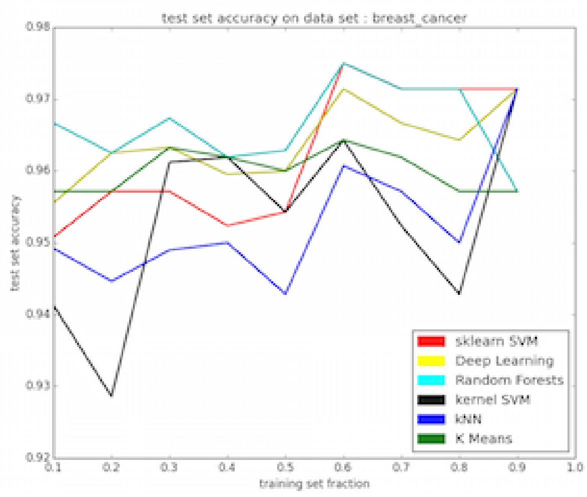


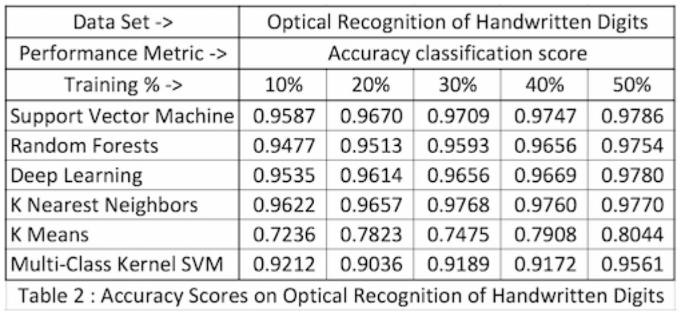
Fig. 11. Accuracy vs training percentage - breast cancer data

B. *Optical recognition of handwritten digits – original*

As part of this dataset, we are using the optdigits.tra.txt as our training data file. The first 64 column vectors are used as the X values, while the last column is used as the concept class.

C. *Forest type mapping*

We are using the training.csv file as the training data file. Since the concept classes in this data set are alphabets, we map them to numeric values, 0,1,2,3 in the numeric_class.py file. Moreover, there is wide variation in the range of the data for the different features in this data set. While some



Data Set ->	Optical Recognition of Handwritten Digits				
Performance Metric ->	Accuracy classification score				
Training % ->	10%	20%	30%	40%	50%
Support Vector Machine	0.9587	0.9670	0.9709	0.9747	0.9786
Random Forests	0.9477	0.9513	0.9593	0.9656	0.9754
Deep Learning	0.9535	0.9614	0.9656	0.9669	0.9780
K Nearest Neighbors	0.9622	0.9657	0.9768	0.9760	0.9770
K Means	0.7236	0.7823	0.7475	0.7908	0.8044
Multi-Class Kernel SVM	0.9212	0.9036	0.9189	0.9172	0.9561

Table 2 : Accuracy Scores on Optical Recognition of Handwritten Digits

Fig. 12. Table 2 : Accuracy Scores on Optical Recognition of Handwritten Digits

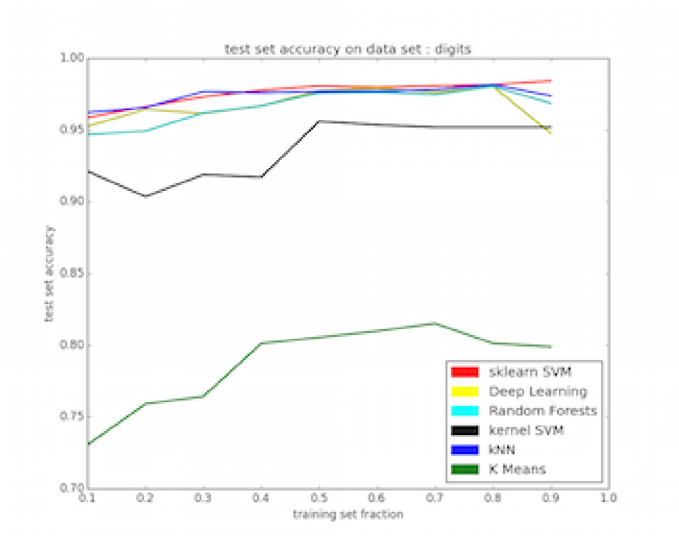
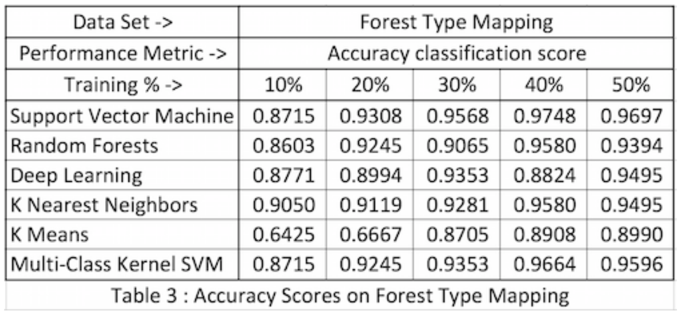


Fig. 13. Accuracy vs training percentage - digits data

features have values ranging from the lower negative to high positives, other features just have higher positive values, with no negative values. So, prior to using the data to train the SVM and Random Forest classifiers, we normalize it using the equation x_norm = (x – mean(column)) / (max(column) – min (column)). Doing so, the accuracy increases from lower 40's to high 90's.



Data Set ->	Forest Type Mapping				
Performance Metric ->	Accuracy classification score				
Training % ->	10%	20%	30%	40%	50%
Support Vector Machine	0.8715	0.9308	0.9568	0.9748	0.9697
Random Forests	0.8603	0.9245	0.9065	0.9580	0.9394
Deep Learning	0.8771	0.8994	0.9353	0.8824	0.9495
K Nearest Neighbors	0.9050	0.9119	0.9281	0.9580	0.9495
K Means	0.6425	0.6667	0.8705	0.8908	0.8990
Multi-Class Kernel SVM	0.8715	0.9245	0.9353	0.9664	0.9596

Table 3 : Accuracy Scores on Forest Type Mapping

Fig. 14. Table 3 : Accuracy Scores on Forest Type Mapping

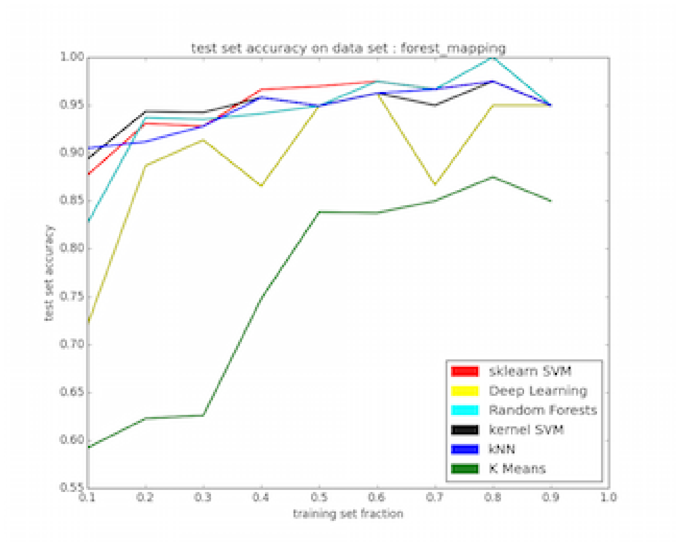


Fig. 15. Accuracy vs training percentage - forest mapping data

VI. Individual Contributions

Each team member worked on data set preparation and the document we are submitting.

A. Jennifer Cheung

Wrote the testing rig and implementation of k-means and k-nearest neighbor.

B. Joshua Kirstein

Wrote the custom multi-class kernel SVM implementation as well as tested it.

C. Abhishek Kumar

Wrote the testing rig and implementation of neural networks using scikit NN.

D. Abhishek Mohanty

Wrote the testing rig and implementation of random forests using scikit.

E. Abhinav Rathi

Wrote the testing rig and implementation of SVM using scikit.