Report: Assignment 1

EE769: Introduction to Machine Learning

Prof: Amit Sethi

Prepared by: Megh Shukla, MTech 1st Year, CSRE

173310008

Introduction:

The problem statement involves classification of the Sale Status of the house. Based on various parameters such as Garage, Grade of Living area, Lot area, we have to classify if the house is sold into 3 categories; 1) Not Sold 2) Sold Slow 3) Sold Fast.

Dataset:

The dataset has 31 features, with the final column as labels of the Sale Status for the training set. Features are both, quantitative and nominal in nature. Also, the dataset has missing values which needed to be dealt with. Another observation we have drawn from the dataset is the skewed number of labels. There are only 9 'Not Sold' labels in the training data, with a roughly even split between 'Sold Slow' and 'Sold Fast'. Hence the data has a severe imbalance in classes. The data is given in an .csv file format. It also appears that the training and testing data have been split evenly from the master data set.

Programming:

Code for the classification is written in Python 3, in the Enthought Canopy Express environment. Libraries used are : 1) numpy 2) matplotlib 3) sklearn 4) scipy (to compute mean median mode) 5) pickle 6) pandas

Numpy is used for scientific computation involving arrays, matrices etc. Matplotlib is a data visualization library which is used to plot hyperparameter tuning graphs. Sklearn is imported to use Machine learning algorithms such as SVM, Random Forests, Neural Networks. We also use sklearn library for hyperparameter tuning. Scipy is used in the code for statistics computation for a particular column in the input data. Pickle is used to store objects such as models trained from the classification algorithm, auxiliary data needed in transforming the input data set (eg : PCA eigenvectors). Pandas is a library which helps in data analysis and manipulation of dataframes.

Organization of the submission:

Submission has 3 folders: Hyperparameter Data, Models and Prediction. The first folder contains analysis of tuning hyperparameter data for various classifiers. Models include pickled files of classifiers and their principal components, if

applicable. The folder also contains pickle files for data manipulation of the input dataset. The submission has 2 '.py' files, test.py and train.py according to the guidelines laid down. I have also attached gt.csv, trainSold.csv and testSold.csv

PLEASE NOTE: The gt.csv given online has columns labelled 'Id' and 'Sold Fast'. I assumed the columns were meant to be 'Id' and 'SaleStatus' (as given in train and test.py). Since my data processing is done by column names, an error will be thrown with the current gt.csv with 'sold fast' as column name. Requesting you to change the column name to 'SaleStatus' in the true gt.csv

Brief algorithm of the train.py:

1) Import the needed libraries
2) Read in both, trainSold.csv and testSold.csv (testSold needed to compute PCA over the entire InputSet.)
3) Normalize the quantitative variables over trainSold.csv and testSold.csv; substitute NA values with the mode of that feature/variable.
4) Conversion of Nominal variables into numerical values
5) Separating the Domain and labels from the Input Set.
6) Tune hyperparameters for Linear Discriminant Analysis, Support Vector Machine, Random Forest and Neural Networks, and plot graphs.
7) Store the models and data processing values using pickle

Brief algorithm of the test.py:

1) Import the needed libraries
2) Read in testSold.csv
3) Normalize the variables using parameters obtained from train.py
4) Convert nominal to numerical values of variables according to labelling form train.py
5) Perform classification prediction on the post processed data using LDA, SVM, Random Forest, Neural Networks; store it as out_<classifier>.csv
6) Check accuracy of each model, display argmax(accuracy) , max(accuracy)

Pre-processing:

The data in it's raw form is unsuitable for machine learning applications. There is need to normalize the data, which has been implemented by subtracting the column i^{th} value with column's mean value, and by dividing with (Max-Min) values for that column. Thus we normalize the values in the column from range -1 to 1.

Three strategies could be employed to treat nominal variables.

1) Nominal to Ordinal
2) Nominal to One-hot
3) Nominal to Binary string columns

The first strategy employs simply labelling the nominal variables with integer values. For example, in the column 'Neighborhood', we could label values 'CollgCr' as 0, 'Veenker' as 1 so on arbitrarily. However, this strategy has fundamental flaws, as we do not know in which order to rank the labels. This leads to relatively poor accuracy in classification.

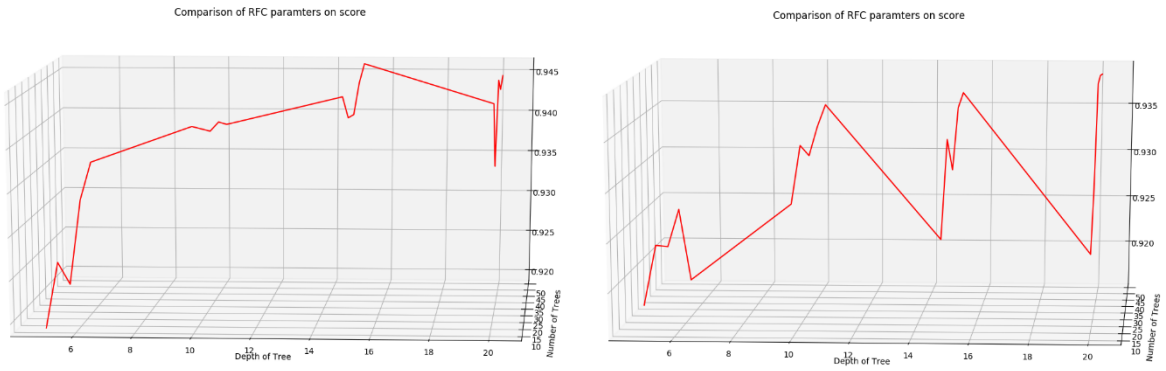
The second strategy, Nominal to one-hot includes having a separate column dedicated to each label in the parent columns. Continuing the above example, CollgCr will have one dedicated column, Veenker will have another column and so on. Hence, a 1 in such a column will indicate presence of that label, 0 will indicate absence. This appears to be more logical to the process of classification, and is proved so by the accuracy measure, which is highest among all the strategies. The drawback of this techniques is the number of features will increase exponentially, hence increasing dimensionality of the input dataset.

The third strategy employs an interesting blend of the first two. The nominal variables are initially numbered as in Nominal to Ordinal conversion. Then, these numbers corresponding to the labels are converted into a binary string. Each position of the string (1s,2s,4s,8s) is represented in a separate column, similar to Nominal to one-hot. Example, 'CollgCr' has value 3, 'Veenker' has value 4. Converting to binary, 'CollgCr': 011, 'Veenker': 100. Since string has 3 digits, there will be 3 child columns for the (1s,2s,4s) position. We then fill in the child columns with the appropriate digit. More clarification can be obtained by opening the ModifiedIP.csv file in the submission. While the logic behind this technique is hazy similar to Nominal to Ordinal, the strategy combines the usefulness of One-Hot conversion in giving better accuracy. It's primary advantage over One-hot is the decreased number of child columns that arise; if there are 'n' labels in the feature, One-hot will give rise to 'n' columns, Binary will give rise to $\log_2(n)+1$ columns, drastically decreasing the dimensionality of the dataset.

For the given dataset, the strategy that worked best was One-hot, since the number of features after one-hot were 106, a manageable amount considering input data has 1400 samples.

Principal Component Analysis:

Principal component analysis is essentially a dimensionality reduction technique which is meant to capture the variance of dataset into fewer dimensions. While PCA may not be essential for the given dataset, certain classifiers such as Random Forest and LDA benefit from the inclusion of PCA. Primarily, PCA might help in reducing overfitting as is seen in the case of Random Forest evidently.



The graph on the left indicates cross-validation score plotted against the depth of the tree in random forest; without PCA as a pre-processing tool. The other plot shows the behaviour of the same classifier with PCA to pre-process the data. It is commonly known that greater the depth of the tree, more likely is the random forest to overfit. This is seen in the first plot, with the performance of the classifier having a slight increasing trend as the depth is increasing, a likely indicator of overfitting. However, with the use of Principal Component Transform, we find that performance measure is constant for different depths of the Tree, with the number of trees in the forest deciding the cross-validation score.

Apart from combatting overfitting, PCA has been included to make the code more versatile, such that it can handle data with larger dimensionality using PCA as a dimensionality reduction tool.

Feature Selection:

High number of features has a tendency to confuse the classifier. Hence there is a need to find optimal features that describe the dataset. The code makes used of Recursive Feature Elimination to obtain the best features that describe the class labels accurately. At each iteration, a small segment of the features are eliminated (either a percent or a small integer), with it's effect on scores being measured. A cross validation approach is used to measure the scores of the classifier. The algorithm terminates with features ranked from best to worst according to suitability for the algorithm. Hence, the input data set is modified to contain those features as specified by RFE.

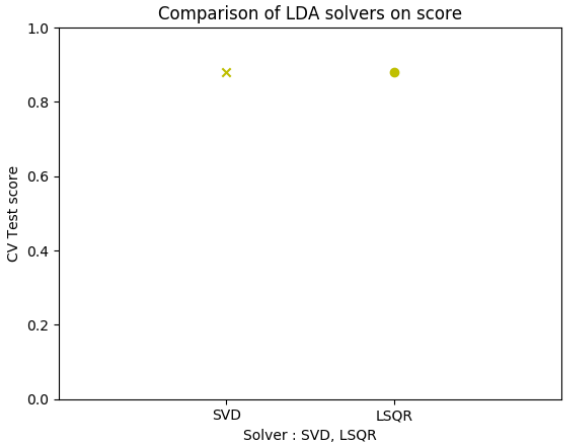
Imbalanced Classes:

I did not choose a complicated method to deal with imbalanced class, my approach involving stratified K fold division of data with all classes balanced in each of the K divisions. Hence cross validation had same ratios of classes in all divisions of the data. I passed 'balanced' keyword to the class_weight hyperparameter to implement this division.

Hyperparameter Tuning:

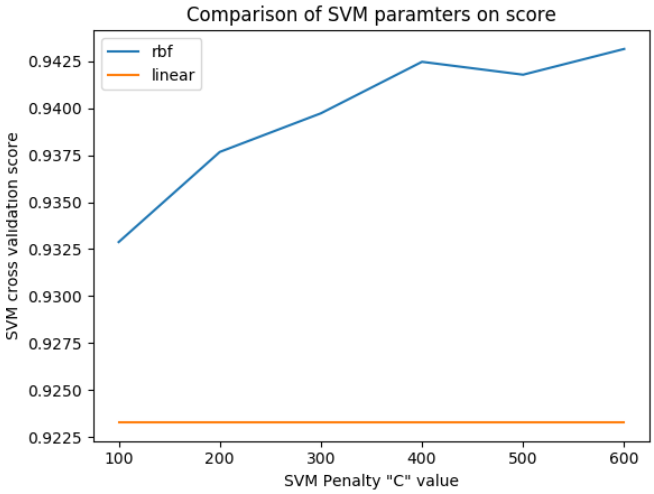
Hyperparameters are those parameters that are decided before the classification algorithm begins learning. These include slack penalty for SVM, number of trees in Random Forests, structure of hidden layers in the neural network etc. An optimal combination of hyperparameters is required to control overfitting at the same time prevent underfitting. Hyperparameters could also be used to reduce computational time of the algorithm. Hyperparameter tuning is easily facilitated by using Grid Search, which makes all possible combinations given the various hyperparameters and compares them using cross validation score. While hyperparameter tuning maybe time consuming, it needs to be performed only once before the learning algorithms operate on the dataset.

LDA:



Linear Discriminant Analysis assumes that each class has the same covariance, and that they are linearly separable. Since LDA has limited number of hyperparameters of importance, I compared the two solvers of LDA; Singular Value Decomposition and Least Square. Eigen solver could not be used as the input data showed collinearity. The hyperparameter tuning graph showed that the 2 solvers have comparable accuracy on the dataset.

SVM:

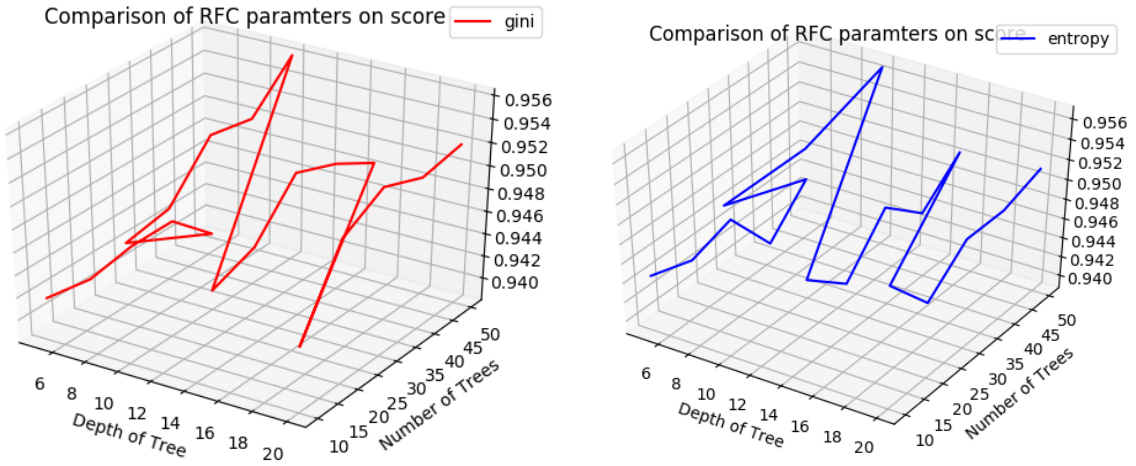


Support Vector Machine hyperparameters that were evaluated were the kernel and the slack penalty ‘C’. SVM using Radial Basis Function kernel becomes a non-linear classifier. These parameters were evaluated using grid search over kernel and penalty. The increasing cross-validation score with increase in slack penalty could be (though not necessarily) an indication of overfitting. Another interesting point of observation is that the classification problem appears to be a non-linear one, with LDA and linear SVM doing poorly compared to their non-

linear counterparts such as QDA and RBF SVM. The cross-validation scores of non-linear models are consistently higher than their linear counterparts.

From the given graph, I have chosen the kernel as 'RBF' with C=100. I have chosen a lower value of C to increase generalization of the algorithm over the dataset, so that it performs better on the testing data set than those with poor regularization (a higher C). A higher value of C might not necessarily perform well on the testing dataset.

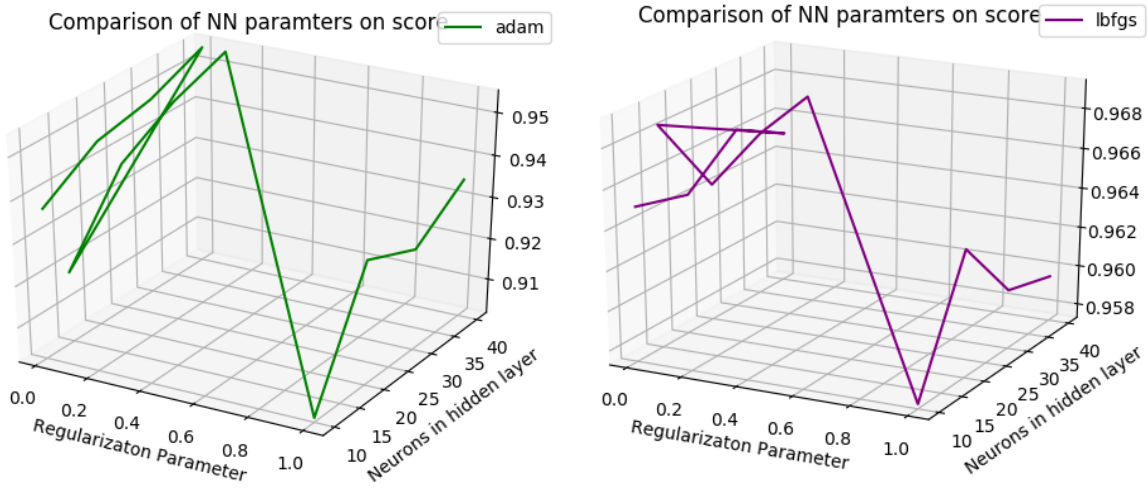
Random Forest:



The random forest algorithm was tested for 3 hyperparameters majorly, the decision: gini, entropy, depth of tree and number of trees in the random forest. We have already discussed the role of PCA in random forest. Gini criteria is meant for binary classification, which is extended to multiple classes using the one vs all classification. Entropy facilitates multi class classification as the cross-entropy function easily extends to multiple classes. We observe that the cross validation score slightly increases as the number of trees in the forest increase. Limiting the depth of each individual tree in the forest can restrict overfitting.

Since this is a multiclass classification problem, we use entropy function. A tree – depth of 10 and 21 trees in the forest is chosen to prevent overfitting and quicken computations of the algorithm.

Neural Network:



Activation function: Rectified Linear Units preferred over Sigmoid

https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions

Multi-layer perceptrons hyperparameters tuned were: alpha (regularization term), neurons in hidden layer and gradient descent approaches of adam and lbfgs. Having more than one hidden layer was not deemed necessity for the problem. Multiple layers risk overfitting and increased computation time. The evaluation of regularization parameters was carried out for 0.01,0.1,1,10. Alpha=10 lead to underfitting, whereas a very low value of alpha tended to overfit the data. The effect of increased hidden neurons is demonstrated, with a slight increase in the CV score as neurons increase. An attempt was made to generalize the model by picking alpha=1 and hidden neurons as 10. (Input features is in the range 5-8 post RFE). We see the performance of lbfgs to be better than adam since lbfgs converges quicker than adam. Hence implementing lbfgs is computationally faster.

Choice of classifier:

While all of the classifiers used gave CV accuracies better than .85, non-linear classifiers such as Random Forests, RBF SVM outperformed linear classifiers such as LDA and Linear SVM. Among the non-linear models, I would choose Random Forests over RBF SVM and Neural Networks. Neural networks, though gave higher scores in cross-validation, is more prone to overfitting than Random forest. Random forests are a powerful algorithm due to the random selection of feature subset for each tree, and random subset of the Input Set for each tree. Thus

this stochastical procedure eliminates bias of any kinds. Combined with PCA, random forests are less likely to overfit. Voting among the trees eliminates outlier data subset (trees containing lots of outliers) from influencing the collective decision of the classifier. In comparison to SVM, it gives superior results, though is computationally expensive. RF allows for more flexible decision boundaries than SVM.

To summarize,

Things that worked:

1. Random forest, Neural Network, SVM (kernel=rbf)
2. PCA to prevent overfitting
3. One-hot approach of converting Nominal variables

Things that didn't work:

1. Linear SVM, LDA
2. PCA for dimensionality reduction (limited features in dataset)
3. Nominal to Ordinal, Nominal to Binary (large number of samples that could handle all the features produced by One-Hot)

Acknowledgement:

1. Professor Alok Porwal, CSRE: For simplifying some of the learning algorithms, and hearing my theories regarding the results. (I did not ask solutions to the problems posed in this assignment to sir!)
2. Professor Amit Sethi, EE
3. Teaching Assistants, EE769