Report - ML Contest

Shreyas Chaudhari - EE15B019
Aravinth Muthu - CS15B004

November 12, 2017

Introduction

Report for the Machine Learning contest hosted on Kaggle.

Understanding the data

The given dataset has 2600 features, with 9501 training samples and 29 classes. A few salient points about the data that clearly stand out are:

- The missing data is only from the first 500 features of both the test and train data. All other columns (features) have 0 missing samples
- The percentage of missing data is very high (around 40% in almost all features)
- The standard deviation of features the does not cross 0.35 for any feature

Based on these characteristics of data, we have selected our data pre-processing methods.

Data pre-processing

Imputation:

Most of the common imputation methods like mean imputation and knn imputation did not give high scores, as expected.

- Mean imputation is not expected to give good results due to the high fraction of missing data in every column.
- Class-wise imputaions methods also performed badly as the domain of the test data is different from the domain of the train data.
- K-nearest neighbour imputation faces the issue that the 'nearest columns' for most of the feautres are themselves missing/imputed columns.

One method that showed better results with the above two imputation methods was class-wise imputation of the training data. But since that demands for the class labels to be known beforehand, thats same cannot be used on the test dataset, leading to imbalance if 'good' imputation for training but a 'bad' imputation for testing. Hence we shifted focus to methods other than classwise imputation.

Upon further reading, we found two algorithms that work well on data that has a large percentage of missing values - Iterative SVD imputation and MICE imputation.

- Iterative SVD: The method is based on singular value decomposition. It performs worse than classwise-knn.

- MICE: Multivariate imputation by chained equations (MICE), sometimes called "fully conditional specification" or "sequential regression multiple imputation" has emerged in the statistical literature as one principled method of addressing missing data. Creating multiple imputations, as opposed to single imputations, accounts for the statistical uncertainty in the imputations. In addition, the chained equations approach is very flexible and can handle variables of varying types (e.g., continuous or binary) as well as complexities such as bounds or survey skip patterns.
  MICE operates under the assumption that given the variables used in the imputation procedure, the missing data are Missing At Random (MAR), which means that the probability that a value is missing depends only on observed values and not on unobserved values. This assumption is suitable for the given dataset and therefore shows good results. For further reading about the implementation of MICE, refer to this paper.

To benchmark all the above imputation methods, logistic regression was used. A test train split of 80-20 for cross validation showed good results for knnimputation and MICE imputation, with MICE being the better one. knn imputation done classwise did increase the cross validation score, but did not perform well on the submitted test data due to the above mentioned reason.

In the case of MICE imputation, various variations were tried before we fixed on the final form. The MICE imputer takes two parameter - *number of imputation* and *number of nearest columns* to be considered. Benchmarking over logreg over a few parameters the final parameters fixed on gave -

```
n_imputations=15
n_nearest_columns=10
```

which also shows that larger number of columns considered/larger number of imputations is not necessarily good for the evaluation metric.

The image below taken from one of the references mentioned below shows the variation of different kinds of error with varying amounts of missing data (randomly removed) of different imputation methods. That further reinforces the choice of MICE as an imputation method.

Figure 1: *Imputation method comparison*

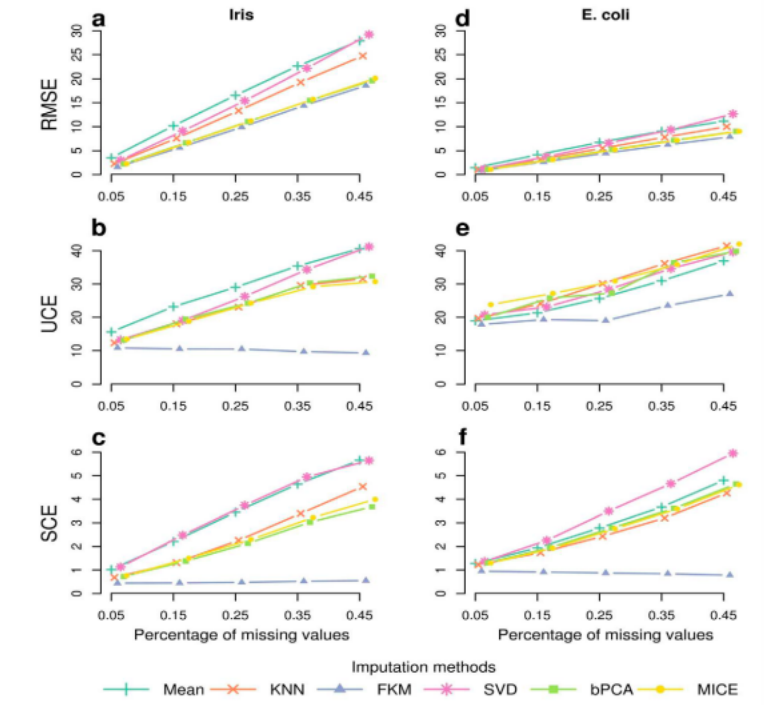


Figure 3: Evolution of RMSE, UCE and SCE with a varying percentage of missing values in the small datasets: Iris (a, b, c) and *E. coli* (d, e, f).

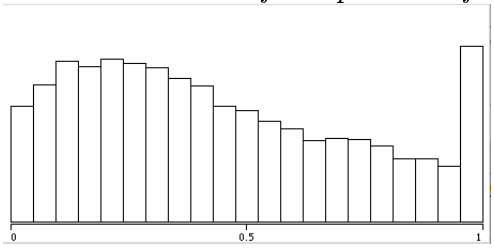
Note: While imputing, the test and the train data are clubbed together so that the same fit is applied for train and test, and values are filled in accordingly. This method gave very good results.

Observation: If the first 500 features were ignored and any classifier was run on the remaining 2100, that gave a better score than considering the entire imputed dataset. To account for the same, feature reduction method was changed accordingly, as described in the following subsection.

Changing the input data:

As can be seen by the distribution of values in a given feature shown below, the last bin of values near one shoots up and has a large bin count. And as can be seen by further visualisations, these values do turn out to be a large number of 1's - as is also shown by the feature correlation figure (Figure 3). We felt that all samples above or equal to one have been recorded as one in this dataset.

Figure 2: *Distribution of samples in a feature*



Ideally, (no comments on practicality) a probability distribution should be fit onto this distribution and values greater than one should be placed at the 'correct' positions, which will be determined by the other features of that sample. Beacuse we do not know of any heuristic to find the 'correct' place to replace the current 1 with, we have not attempted to do so. Instead, we replaced all 1's with *nan*s and ran imputation, the resultant dataset giving a lower fscore, as expected, since the information we have in the form of 1 is lost as some value between 0 to 1.

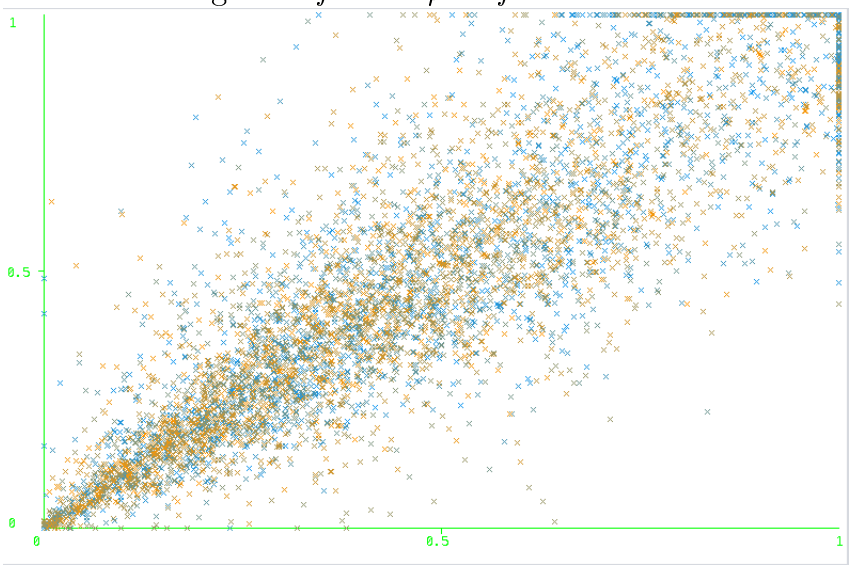
Feature reduction

The feature reduction method being employed is Principal Component Analysis. Ths was decided due to the following reasons:

- VarianceThreshold - choose features above a certain threshold variance value (to avoid almost constant features). In the given dataset, every feature is in the vicinity of 25% standard deviation, with a few ranging from 24% to 33%. Therefore, firstly the variance of every features is too low to successfully apply variance thresholding, and secondly, due to most features having similiar variances the method did not give good results, as expected.

- Feature Agglomeration - As analysed, some features have a very high degree of correlation. An example is shown below:

Figure 3: *feature 4 vs. feature 2535*



The plot also depicts the aforementioned saturation of many values at 1. This method showed highly promising results on cross validation data split of 80-20 with fscores going up to 42%, but the highest submission score came up to a mere 39.8%, which probably indicates overfitting on the model's part. This is one of the models that has been used in the second layer of the final ensemble.

- Principal Component Analysis - uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables. The difference being that in the previous approach, the 'direction of the dimensions' are retained, only some dimensions are dropped, whereas PCA determines a new set of directions for the dimension vectors. The unconventional method of feature reduction using PCA is described below.

Unique technique: Based on the observation stated above, that a classification by ignoring the first 500 features gave a sharp increase in evaluation score, in case of PCA, PCA was applied separately to the first 500 features, and the remaining 2100. A gridsearch over the two parameters (number of features of the first 500 and the next 2100) gave the optimal value as being 200 from the first 500 and 1800 form the next 2100, adding up to a total of 2000 features.

Classifiers

The classifiers tried out on the data are:

- LDA
- Logistic regression
- XGboost
- Adaboost
- Random forest
- SVMs
- Neural networks

Starting off with logreg as a benchmark, LDA gave very poor results (due to its strong assumptions of data distribution). The best results were given by SVMs.

Observation: Bagging reduces the fscore, for any given model. Since bagging tackles high variance/overfitting of data, this implies that the dataset has an already low variance and bagging will not boost the performance. Therefore, the complementary choice to implement boosting instead of bagging led to to the usage of xgboost and adaboost. Touted to be the wininng algorithm for Machine Learning contests, even xgboost did not give good results which indicates that there is no high bias existing either.

***Support Vector Machines*:**

With the Kernels coming to excellent use, SVMs gave progressivley higher scores with better and better data pre-processing techniques. As mentioned above, the 'best' dataset with 200 features from the first 500 and 1800 from the next 2100 was used to run a gridsearch to find the best parameters on cross validation for *sigmoid* and *rbf* kernels. Below are the results for both the grid searches:

Figure 4: *rbf kernel gridsearch*

Gridsearch over RBF parameters

	C=0.1	C=1	C=10	C=100
gamma= 0.001	0.175,	0.30759494	0.35981013	0.36139241
gamma=0.01	0.18639241	0.37753165	0.40443038	0.40443038
gamma=0.1	0.05094937	0.06455696	0.07120253	0.07120253

Figure 5: *sigmoid kernel gridsearch*

Gridsearch over Sigmoid parameters

	C=0.1	C=1	C=10	C=100
gamma= 0.001	0.10474684	0.18702532	0.19208861	0.1778481
gamma=0.01	0.05126582	0.03132911	0.025	0.04050633
gamma=0.1	0.05094937	0.05094937	0.05031646	0.04936709

which led us to fixing the parameters of SVMs at *c =10 and gamma = 0.01* and the kernel as *rbf.* The other parameter that could have varied a lot of results was the choice betweek one versus all and one versus rest. Given below are the class sizes:

Table 1: *Class sizes*

Class	Size	Class	Size	Class	Size	Class	Size
1	403	9	277	17	480	25	346
2	311	10	379	18	382	26	150
3	394	11	333	19	474	27	340
4	153	12	254	20	220	28	351
5	304	13	130	21	468	29	176
6	472	14	271	22	425		
7	252	15	258	23	436		
8	243	16	336	24	483		

which indicates that one vs rest may lead to class imbalance while evaluating its hyperplanes. But that is not the case as both are seen to give similar cross validation scores, with one vs one just taking a longer computational time as expected.

Each of the above combinations was tried for differently pre-preprocessed data, giving the *best submission model* with the MICE imputed (200+1800) PCA feature reduced dataset.

The other form of feature reduction - Feature Agglomeration - gave promising results on cross validation (42% fscore) but the final submission coming only up to 39.8%, implying overfitting of the train and test.

Neural Networks:

Neural networks: To fit high dimensional data, neural networks are a very good choice. We explored deep networks with different depths and neuron counts. We also varied the minibatch size, learning rate and optimiser functions. The three models out of all that we tried that has given us the best insights are explained below.

- **Model1:**
  Input: Hidden layers: 2000,1000,500.
  Learning rates: 0.001, 0.0001
  Optimiser: Adam
  Observation: The model overfit early on giving a max validation score of 33.39

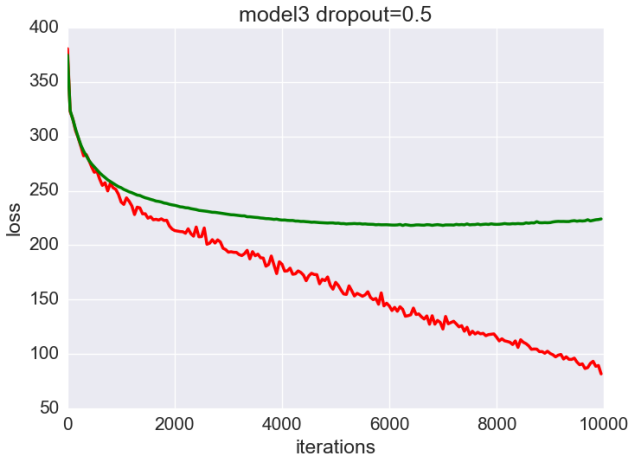
- **Model2:**
  Input:
  Hidden layers: 1500,500,200
  Learning rates: 0.001, 0.0001
  Optimiser: Adam
  minibatch size: 1000
  Observation: The model again overfit early on giving a max validation score of 33.39 The training loss decreae was much smoother than with smaller minibatch sizes previously.

- **Model3:**
  Input: 2000D (200 best features(PCA) of first 500 from mice imputation, first 1800(PCA) from the rest of 2100 features)

Hidden layers: 800, 300
Dropout (keep_probab) values tried: 0.9, 0.7, 0.5, 0.2
Learning rates: 0.001 with step descent after 10000 iterations
Optimiser: Adam
Observation: We tried to introduce batch_norm after each layer, but the training almost saturated very early making incremental updates, hence we removed it. The learning was much faster with keep_probab = 0.9, but the test loss hit a minimum at test accuracy = 34.78 The best validation accuracy of 37.4 was obtained with keep_probab = 0.5 With keep_probab = 0.2, the model was unable to overfit completely, the training accuracy reached only a maximum of 0.83 after 100000 iterations.

The training and test loss of the first 10000 iterations of Model 3 with dropout (*0.5*) is shown below:

Figure 6: *green - test loss; red - train loss*



The evaluation metrics are understandably low due to overfitting by the above models, which can be depicted by the following TSNE plots:

Figure 7: *Initial TSNE plot*

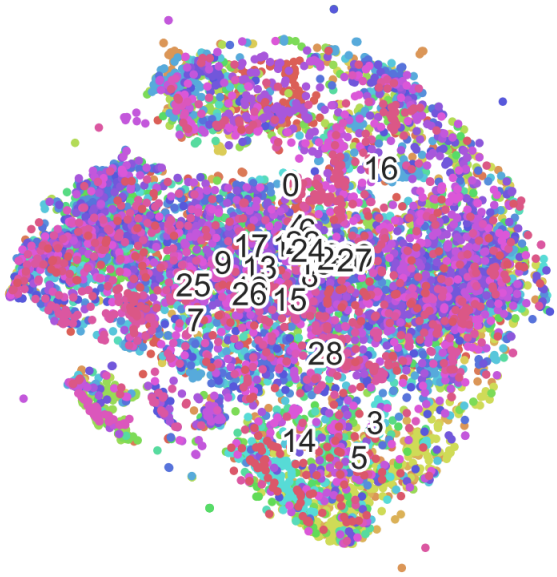
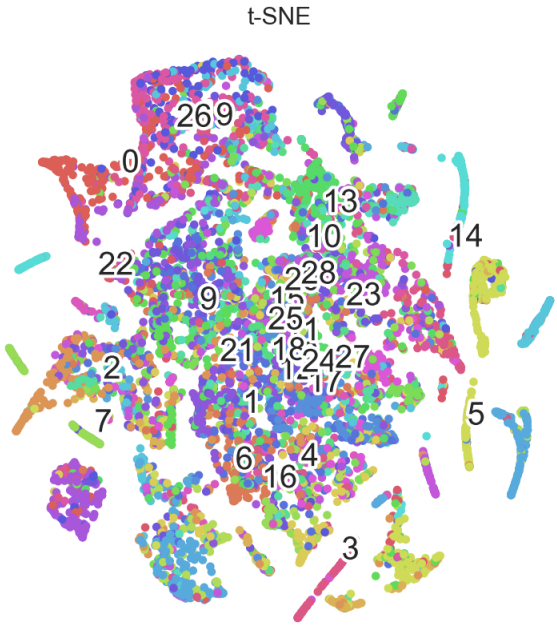


Figure 8: *TSNE plot - output of neural network*



As can be seen, the data points that belong to the same class are separated out into smaller separated clusters which is due to overfitting.

XGBoost:

Due to bagging not giving a good result, the counterpart boosting was tried. The tunable parameters are: *eta, max_depth, number of iterations.* With a cross validation score of 34%, this model was used in the final weighted voting ensemble.

The two kinds of parameter variations gave the following results:

- *max_depth = 2 (stub), max_iterations = 200* - gave 34.7% fscore on validation
- *max_depth =* 1*(root), max_iterations = 300* - training fscore saturated at 62%, while validation fscore gave 27%
- *max_depth = 10, max_iterations = 50* - gave 0.98 fscore on train and 27.2% on test (implying overfitting)
- *early_stopping* - was set to 1/10th of the max_iterations which kept track of the increment in validation score and terminates when the model does not improve over the specified iterations

Random forest:

Was implemented to try and overfit the data, since bagging gave no good results. Increasing the number of estimators and depth of the model just saturated the fscore at around 31%, even with variation in the number of features being randomly sampled by each tree.

K-nearest neighbours:

We ran KNNs over k = 2 to k = 30. There wasn't a significant increase in score with increasing k. All values of k gave less than 24% fscore.

Ensemble

The idea of ensembling classifiers is to use outputs of weak classifiers and cumulatively generate a better output. We used two layers of ensembling the classifiers. In the first layer, we used the outputs from:

1. Neural networks:
   i. With dropouts - (3 models)
   ii. Without dropouts - (1 model)
2. SVM: Every model used random 80% of the data, since bagging caused reduction in performance of the model. Since we were fairly confident about the hyperparameters to get the best SVM, we had to sample such distributions to bring in variation.
   i. rbf: c =10, gamma = 0.01 on PCA reduced dataset (20 models)
3. XGBoost

Three weak models each with a score of around 35%, form the first layer of ensemble.

The output of the first layer of ensemble was then used for the second layer, in which the other participant classifers were given weighted vote based on their fscores:

1. SVM - C=10, gamma=0.01, on MICE imputed PCA reduced data (200+1800 as mentioned above) *(the second submitted model with the highest fscrore)*

2. SVM - C=10, gamma = 0.01, on MICE imputed Feature Agglomeration reduced data (data in which correlated features have been clustered together) with *n_clusters = 2000*

3. Output of previous layer

This the the first model that has been submitted as a 'final submission.
The second model submitted is :- SVM - C=10, gamma=0.01, on MICE imputed PCA reduced data (200+1800 as mentioned above)

References

[1] fancyimpute
[2] MICE - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3074241/
[3] iSVD - https://www.ncbi.nlm.nih.gov/pubmed/11395428
[4] MICE - https://www.omicsonline.org/open-access/a-comparison-of-six-methods-for-missing-data-imputation-2155-6180-1000224.pdf
[5] sklearn documentation
[6] tensorflow documentation

Closing Remarks:

We were glad to work on a real Machine Learning contest (with synthetic data) exploring the different classifiers and their uniqueness. We also understood the need for data exploration and analysis, which is just as important as building powerful models. We learnt the importance of having a proper schedule and working in a structured way. We learnt to colloborate online using Git which saved us a lot of time. DISCLAIMER: Everything in this report is true to the core, except the last line. We used Gmail.