CS725: Foundations of Machine Learning
PROJECT REPORT

Analysing and Boosting of ML models for Diabetes prediction

BALBIR SINGH (22M0747) ABHINAV GARG (22M0750)
ABHISHEK DIXIT (22M0761) NIKHIL SHARMA (22M0770)
SHUBH MALHOTRA (22M0780)

November 2022

Introduction

One in every eleven people in India suffers from diabetes and Diabetes has lead to the death of 700,000 people in 2020 alone. Also, around 10.5% of the world's population suffers from some form of diabetes and this number is increasing rapidly. Hence, it is necessary to predict and counter diabetes as early as possible, otherwise it becomes chronic and the chances of death increase. It is very hard to find the actual cause of diabetes and hence prediction based on multiple features is a good approach. This became a motivation for our project.

For our project, we are training various Machine learning models like Naive Bayes, Logistic Regression, Support vector machines, Decision Tree, Random forest and Neural network and using them to predict the chances of suffering from diabetes given other relevant details like Age, BP, Cholesterol levels, BMI etc.

To increase the accuracy of the predictions even more, we have also implemented a **Hybrid Ensemble Boosting** algorithm which uses the above mentioned classifiers to create and train a stronger classifier with an expected accuracy better then its constituent models.

Problem Statement

Our dataset[1] contains information about 229781 unique people and includes 21 pieces of physical as well as general information about them. To do effective diabetes prediction, we have trained our models for a classification task to detect correlation between these 21 features and the target variable (Diabetes_binary) i.e. if the person is suffering from diabetes or not.

1 General Data Preprocessing

1. Checked for missing values. None found.
2. Applied Exploratory data analysis and found extreme skew in the classes of output variable. Label 0 = No diabetes = 213703, Label 1 = Prediabetes = 4631, Label 2 = Diabetes = 35346
3. Combined Prediabetes and Diabetes classes in the target variable so that when undersampling applied, too much data is not lost. Hence, converted the multi-class classification problem to a binary classification problem.

4. Plotted a correlation plot between input variables and the target variable and dropped the variables with near zero correlation. 'Fruits', 'Veggies', 'AnyHealthcare', 'NoDocbcCost', 'Sex' were dropped.

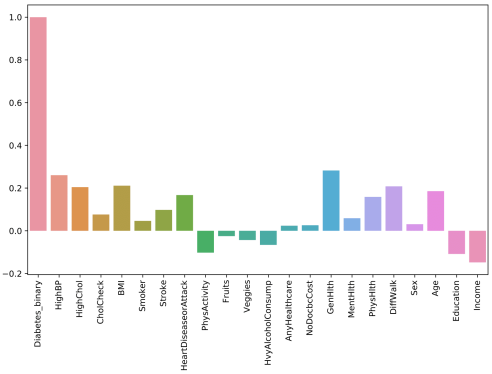

Figure 1: Correlation plot between input variables and target variable

2 Naive Bayes

Naive Bayes[2] is a supervised learning algorithm and is an application of Bayes theorem. It is based on the naive assumption that the input features are conditionally independent of each other. Given a class variable y and independent input features x_1 to x_n

$$P(y|x_1, ..., x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y) \qquad (1)$$

2.1 Gaussian Naive Bayes

The likelihood of features is assumed to be gaussian in GNB:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) \qquad (2)$$

σ_y and μ_y are the standard deviation and mean respectively

- Gaussian Naive Bayes has been used to account for continuous variables in the data set.

Naive bayes operates on the naive assumption of conditional independence between input features. We found that there is high correlation between features like physical health, mental health, income and diffwalk. Features with high correlation have been dropped to account for conditional independence between features.

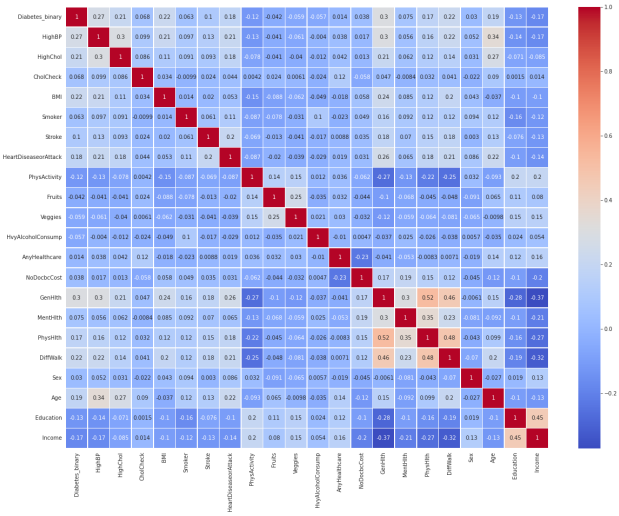

Figure 2: Correlation heatmap between the input variables

2.2 Results

Accuracy achieved on Train set = 78.3755%
Accuracy achieved on Test set = 78.2114%

3 Logistic Regression

As Logistic Regression[3] is a regression based classifier, it gives the probability of the output variable based on the input variables. The prediction function is represented as a sigmoid function of linear combination of the inputs

$$f_w(x) = \sigma(w^T x) = \frac{1}{1 + exp(-w^T x)} \quad (3)$$

We try to choose the weights for which the cross-entropy error function is minimized -

$$w_{LR} = \underset{W}{\operatorname{argmin}} - \sum_{i=1}^{N} \log P(y_i|x_i, w) \quad (4)$$

3.1 Training and Tuning

For this project, we used the Logistic Regression model from sklearn [1] library.

- Converted categorical and continuous input variable to one hot vector columns.
- Used SMOTE oversampling [8] to remove skew from output variable classes.
- Implemented grid search to find the following optimal values for the hyperparameters
  - solver = saga (algorithm used for optimization)
  - C = 0.9 (inverse of the regularization extent)
  - penalty = elasticnet (tells about norm of regularization)
  - l1_ratio = 0.9 (ratio of l1 norm in elasticnet penalty)
  - tol = 1e-3 (tolerance of the stopping criteria)
  - max_iter = 300 (iterations to stop after if not converged)

3.2 Results

Accuracy achieved on Train set = 87.1671%
Accuracy achieved on Test set = 87.2690%

---

4 Support Vector Machines (SVM)

SVM[4] finds the hyperplane that separates the classes of data as widely as possible (i.e. with highest **Margin**), Such Hyperplane is called **Maximum Margin Hyperplane.**

4.1 Preprocessing Steps

- Different Feature scaling[7] methods were applied i.e. MaxAbsScaler, MinMaxScaler, StandardScaler. Out of which **MaxAbsScaler** was found optimal for SVM on our dataset.
- Tried different Undersampling and Oversampling techniques, optimal is **NearMiss** for SVM on our dataset.
- Converted categorical features into multiple features using one-hot encoding. (except for features: 'BMI', 'MentHlth', 'PhysHlth', 'Age', as these features are continuous). One additional advantage of this for SVM is that it performs better for **higher dimensional** data.

4.2 Kernels

The key idea in SVM is **Kernel tricks** and the main task in SVM is to find the right Kernel. SVM can be used for both, linearly separable datapoints and non-linearly separable datapoints using Kernels. Kernel in SVM transforms the features to Higher dimensions. Kernel is used in the dual form of constrained optimization objective functions of SVM to observe similarity relationships between the datapoints.

In our Project we used existing kernel tricks i.e. Linear Kernel, Polynomial Kernel, Radial Basis Function (RBF). **RBF** and Polynomial kernels gives better performance when tuned with proper hyperparameters. Linear Kernels gives accuracy around **0.85%.**

RBF kernel :

$$K(X_1, X_2) = \exp\left(-\frac{||X_1 - X_2||}{2\sigma^2}\right) \quad (5)$$

where (σ) is the variance and **gamma** is:

$$\gamma = \frac{1}{2\sigma^2} \quad (6)$$

4.3 Hyperparameter Tunning

Best kernel and Optimal Hyperparameter values were found by grid search, as mentioned below-

- **C = 20**, C decides that how much importance is given 'to not make a mistake' or error or **slack** (inverse of the **λ** of logistics regression)
- **gamma = 1**, gamma is the parameter of RBF to handle non-linear classification. Higher the value of gamma more the curvature in decision boundaries. gamma is inversely proportional to sigma in RBF functions
- **degree = 3**, degree of the polynomial kernel functions, It is only applicable to polynomial kernel.
- **tol** = 1e-3 (default)

4.4 Observations

* Time taken to train SVM is **very high**.
* In our dataset it gave **higher accuracy** than all the other classifiers.
* SVM perform very well in **small** , **non-linear** and **Higher dimensionality** datasets.

4.5 Results

Accuracy achieved on Train set = **91.3909**%
Accuracy achieved on Test set = **89.7175%**

---

5 Decision Tree

Decision Tree[5] is a simple supervised machine learning algorithm. A decision tree is composed of many decision nodes and leaf nodes. Leaf nodes stores the prediction of the model. Decision nodes denotes the split of the dataset at that node using a specific feature.

5.1 Training process

To train a decision tree we start from the root node. Then make a decision to split the node into two parts using the best feature. Best feature is chosen using a splitting criterion. This process of splitting of tree into subtree is done recursively till we get the optimal depth of the tree.

5.2 Splitting criterion

1. Entropy: It denotes the uncertainty or impurity of the distribution. We first check the entropy of the current node and then that of its child node. More the decrees in entropy from the current node to the child node, the better is the split. This decrease in entropy is known as Information gain.

$$Entropy = -\sum_i p_i * log_2(p_i) \tag{7}$$

$$\text{Information Gain} = \text{Entropy}_{parent} - Entropy_{child} \tag{8}$$

2. Gini Index: It denotes the purity of the distribution. Less the value of Gini, more pure is our distribution. To find the best feature for splitting we split the current node into two child. Then we calculate the Gini Impurity over the both child nodes.

$$Gini = 1 - \sum_i (p_i)^2 \tag{9}$$

* The accuracy was found to be very similar in both so we decided to use the gini index as it requires less computational power.

5.3 Observations

1. Best features were found to be PhysicalHealth at the root node and Income and DiffWalk at the first left and right child node respectively.

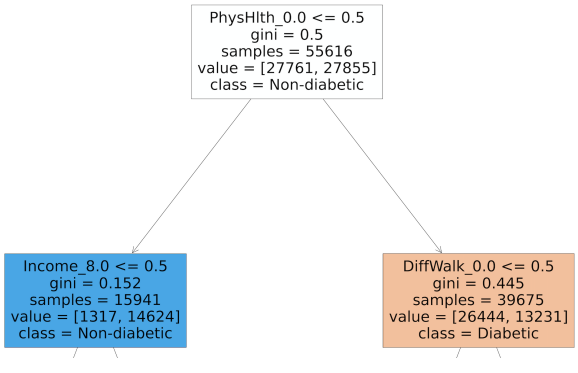

Figure 3: Decision tree split at root node based on gini index

2. Optimal depth was 13 out of the 51 total features. At deeper depth the tree was

getting overfitted and at lower depth the accuracy was very low.

5.4 Results

Accuracy achieved on Train set = 84.3849%
Accuracy achieved on Test set = 82.8996%

---

6 Random Forest

Random forest is an ensemble of decision trees. It helps in overcoming the overfitting of the decision trees. As in this we are creating many trees with a low correlation between each.

To deal with the overfitting it uses a technique known as bagging.

6.1 Bagging

1. Bootstrapping: For each tree the dataset is created by randomly picking the data points from the original dataset with replacement.

   Feature Randomness: To train one decision tree we select a random subset of features. This decreases the correlation between each tree. Otherwise the structure of most of the trees would have been the same.

2. Aggregation: After training each tree, prediction is done overall. The class which gets the majority of votes is chosen as the final prediction.

6.2 Hyperparameters used

1. No of decision trees = 100. If we use a small number of trees then it is very likely that the overall random forest won't be able to use the complete dataset and features.
2. Splitting criterion = Gini Index
3. Depth = 0.2 of the max depth

6.3 Implementation and Observations

1. We have used the Decision Tree library from scikit-learn as a sub-routine to implement the random tree. The accuracy achieved using our implementation and using scikit-learn library of random forest were found to be very similar.
2. Feature randomness: For this, we are not selecting a random subset of the same size but different for each tree. We first randomly chose a fractional value from 0 to 1 and then that fraction of the features are randomly selected to train the tree.
3. We have set the depth to 0.2 of the max depth for each tree , this way the tree doesn't overfit and at the same time we are able to use all the features to train the random forest.

6.4 Results

Accuracy achieved on Train set = 84.0585%
Accuracy achieved on Test set = 83.6927%

---

7 Neural Network

The architecture used to train the Neural Network[6] model consists of 1 input layer, 2 hidden layers and 1 output layer.The hidden layers consists of 32 and 64 neurons respectively.ReLu and sigmoid activation functions are used at hidden layers and output layer respectively.

7.1 Training and Tuning

For this project, used keras framework to train neural network classifier.

- Tried different oversampling and undersampling[9] methods to deal with the skewed data.NearMiss undersampling worked best in case of neural network classifier
- Used Robust Scaling method to deal with the outliers in the dataset.
- The loss function used to train the model is binary crossentropy since it is a binary classification problem.
- Implemented grid search to find the following optimal values for the hyperparameters

– Batch Size = 32

– Learning Rate = 0.001

7.2 Results

Accuracy achieved on Train set = 87.7330%
Accuracy achieved on Test set = 87.0367%

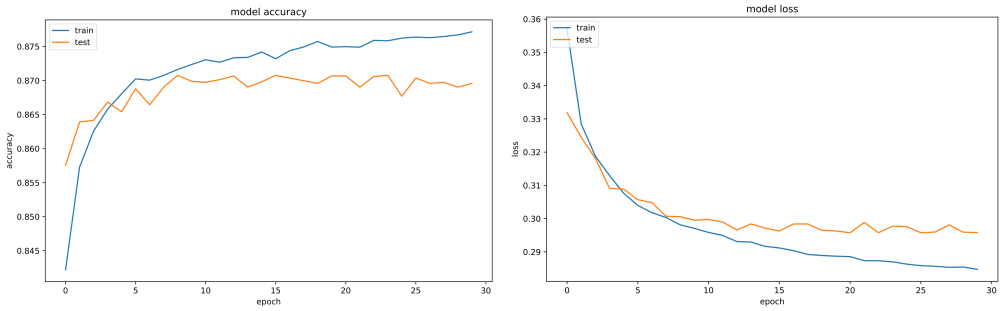

Figure 4: Epochs vs Accuracy (On Left) and Epochs vs Loss (On Right)

8 Boosting

Boosting[10] is a method by which a strong classifier can be trained using a number of weak classifier. The figure below explains how we can train a classifier using boosting.

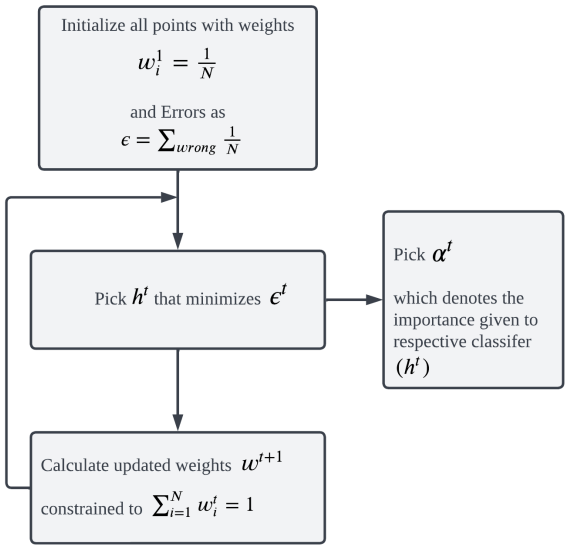

Figure 5: Flow chart of Boosting process

Initialize weights as

$$w_i^1 = \frac{1}{N} \tag{10}$$

At each iteration, error is calculated as the sum of weights of all misclassified points subject to the constraint that sum of weights of all points remains 1. Weights are updated as

$$w_i^{t+1} = \frac{w_i^t}{Z} \times \begin{cases} \sqrt{\frac{\epsilon^t}{1-\epsilon^t}} & \text{Correct} \\ \sqrt{\frac{1-\epsilon^t}{\epsilon^t}} & \text{Wrong} \end{cases} \tag{11}$$

where Z is the normalizing constant

Different classifiers used in the boosting process can be of same class as well as from different classes.

8.1 With Classifiers of Same Class

We used decision stumps classifiers in the boosting process to train a classifier. For this we used the Decision Tree Classifier from sklearn library with a max depth of one in our implementation for boosting.We have used 100 such decision tree stumps to train our classifier.NearMiss undersampling method is used to handle the skewed dataset.

In order to tune the hyperparameters, Grid search is used for each of these decision stumps and we achieved the following results

8.1.1 Results

Accuracy achieved on Train set = 82.8269%
Accuracy achieved on Test set = 82.3125%

8.2 With Classifiers of Different Classes (Hybrid Ensemble)

To train a Hybrid Ensemble model we used the classifiers of different classes, specifically SVM classifier, Neural Network classifier, Logistic Regression classifier,Decision Tree classifier and Naive Bayes classifier. The weights of each of the points were calculated using the formula given at the starting of this section.

To handle the skew in the data samples NearMiss undersampling is used, and grid search is used to tune the hyperparameters for each of the classifier.We achieved the following results.

8.2.1 Results

Accuracy achieved on Train set = 86.2324%
Accuracy achieved on Test set = 85.1947%

8.2.2 Observations

- The accuracy of individual models during the boosting process is lower as compared to their performance when these were trained individually.This could be because of the following reasons.
  - The misclassified points are given more weight and hence the model is trying to be correct for these points.
  - During the individual training process, specific data processing techniques were used for each of the models. However here we are using a generalized preprocessing techniques.
- The accuracy of the boosted model has not increased significantly as compared to individual model performances.This is due to high overlap between misclassified points for different classifiers

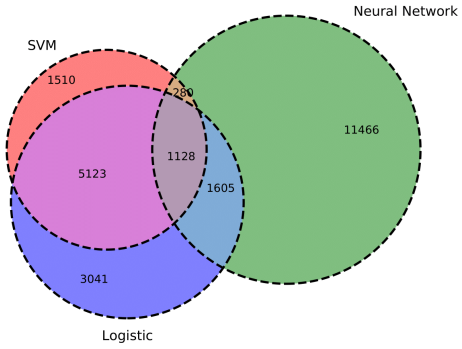

Figure 6: Diagram showing overlap of misclassified points by different classifiers

8.3 Results

S.No.	Model	Initial test accuracy (without preprocessing and hyperparameter tuning)	Final Accuracy (after preprocessing, grid search and hyperparameter tuning)
1	Naive Bayes	0.753587	Train: 0.783755, Test: 0.782114
2	Logistic Regression	0.833040	Train: 0.871671, Test: 0.872690
3	Support Vector Machine	0.827803	Train: 0.913909, Test: 0.897175
4	Decision Trees	0.779932	Train: 0.843849, Test: 0.828996
5	Random Forest	0.842925	Train: 0.840585, Test: 0.836927
6	Neural Network	0.830689	Train: 0.877330, Test: 0.870367
7	Ensemble with Same Class	-	Train: 0.828269, Test: 0.823125
8	Hybrid Ensemble	-	Train: 0.862324, Test: 0.851947

We observed that during individual training of the classifiers SVM gave the best accuracy on this dataset followed by Neural Network and Logistic Regression, also for this dataset dataset hybrid ensemble gave a much better accuracy than ensemble of classifiers of the same class.

Bibliography

[1] Dataset - https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset

[2] Naive Bayes - https://scikit-learn.org/stable/modules/naive$_bayes.html$

[3] Logistic Regression - https://scikit-learn.org/stable/modules/generated/sklearn.linear$_model.$
$LogisticRegression.html$

[4] Support Vector Machines - https://scikit-learn.org/stable/modules/svm.html

[5] Decision Tree - https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[6] Neural Network - https://www.tensorflow.org/api$_docs/python/tf/keras$

[7] Scaling - https://scikit-learn.org/stable/modules/preprocessing.htmlpreprocessing

[8] Oversampling - https://imbalanced-learn.org/dev/references/over$_sampling.html$

[9] Undersampling - https://imbalanced-learn.org/dev/references/under$_sampling.html$

[10] Boosting - https://youtu.be/UHBmv7qCey4

Project Links

- GitHub link to the Project - Github
- Google Colab link to the Project - Colab
- Presentation Link - Google Slides