**Practical Machine Learning - Assignment 2**

**Name : Pavithra Ramasubramanian Ramachandran**

**Student ID : R00183771**

**Abstract**

Building a machine learning model does not just involve collection of data samples and testing the model for prediction metrics. It is important to consider the quantity and quality of data that is fed to the model for training. Typically, more the training data, better the results. However, redundant data in excess too can overfit the model. Similarly, raw data might contain excessive amount of junk data or missing values, etc which may lead to reduction in impact of data that have high certainity in predicting classes. In this assignment, we focus on pre-processing data such that these issues are addressed. Parallely, we also pick models to suit our dataset to build the machine learning model and thus evaluate their performances.

**1 Introduction**

The primary focus of the assignment was to build a machine learning model that classifies data samples into specific classes. By virtue of the scikit-learn package, building machine learning models has become a cakewalk. However, the intent is to obtain predictions with better accuracy keeping in mind several other factors that may bring in bias or variance into the prediction model. To do so, we need to implement a series of pre-processing techniques to the data before feeding it to the model for prediction. The right combination of data pre-processing techniques, model selection and hyperparameter tuning will give a considerably good machine learning model for predicting data classes.

The dataset used in this assignment was downloaded from Kaggle forum[1]. The dataset is about ‘lower back pain symptoms’. It consists of 12 columns of Numerical data and one column which is the class label column. The data samples are classified into 2 classes – Normal and Abnormal based on the 12 columns which signify tilt or angle of the spinal chord or pelvic bones in the body. The below table specifies what exactly each column of the data specifies.

|  |  |  |
| --- | --- | --- |
| Column Name | Column Represents | Data Type |
| Col1 | pelvic\_incidence | Numerical |
| Col2 | pelvic\_tilt | Numerical |
| Col3 | lumbar\_lordosis\_angle | Numerical |
| Col4 | sacral\_slope | Numerical |
| Col5 | pelvic\_radius | Numerical |
| Col6 | degree\_spondylolisthesis | Numerical |
| Col7 | pelvic\_slope | Numerical |
| Col8 | Direct\_tilt | Numerical |
| Col9 | thoracic\_slope | Numerical |
| Col10 | cervical\_tilt | Numerical |
| Col11 | sacrum\_angle | Numerical |
| Col12 | scoliosis\_slope | Numerical |
| Class\_att | Class label | String {Normal,Abnormal} |

The dataset has only 310 rows of data in total which is a very small number. Although the number of data samples is small, there is scope for exploring data pre-processing techniques such as Outlier detection, sampling of data and feature selection. Also, we can take advantage of the size of this data to experiment on machine learning models which take a lot of time to build with hyper parameter tuning.

**2 Methodology**

The methodology followed in the assignment can be broadly sequenced into 3 steps:

* Data pre-processing
* Model Selection and Evaluation of Models
* Hyperparameter tuning and Evaluation

**2.1 Data Pre-Processing**

The data set is first read as a Pandas DataFRame into the Jupyter notebook for further processing. Since data size is just 310 rows with 12+1 columns, we do not split the data into train, validate and test sets. Instead we use the Stratified k-fold cross validation technique to manually train and validate different section of the data. We shall discuss this in detail below. The main steps involed in data pre-processing are as follows:

* Scaling Data
* Outlier Detection
* Dealing with Missing Values
* Feature Selection
* Handling Imabalance

**2.1.1 Scaling Data**

Scaling is an essential pre-processing technique because scaling of data enables us to view data columns from a more generalized perspective in order to perform outlier detection, to plot visualizations, etc. Sometimes, certain machine learning models mandate that the data fed to them has to be scaled. Therefore, implementation of scaling makes model builing easier. There are two methods used to scale data – Normalization and Standardization.

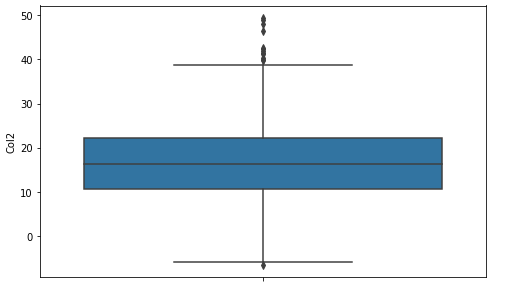
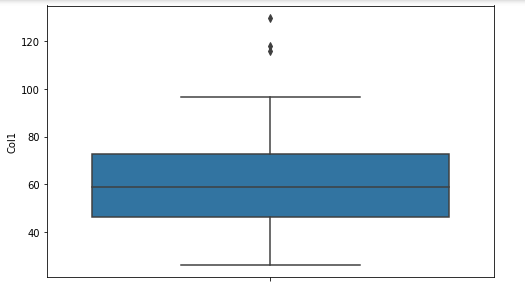
* Normalization : This process scales data between the range 0-1. This means that all data values in all columns will be scaled between 0 and 1.
* Standardization: This process scales the data in such a way that the mean of all the columns will be 0 and standard deviation will be 1.

In this assignment, I have used Standardization technique to scale my data using the StandardScalar() function.

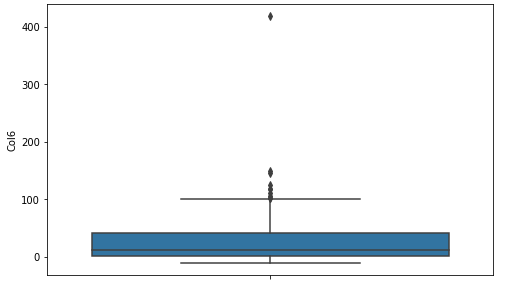
**2.1.2 Outlier Detection**

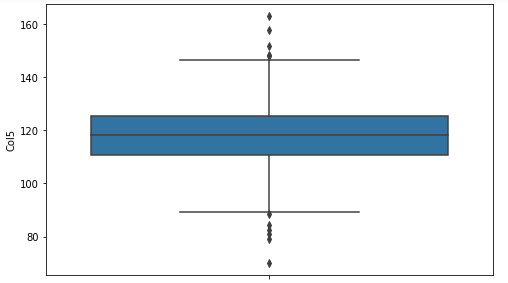
Outliers are data samples that differ significantly from the other data points in the data set. Outliers skew the data distributions and impact the basic statistical measures and can be responsible for the underperformance of certain machine learning models. Outliers maybe representation of some unusual scenarios. Therefore, in order to deal with outlier detection, one needs to have considerable domain knowledge.

Our dataset is about lower back pain symptoms. Clearly, my domain knowldege is zero in this aspect. However, I have handled the outlier scenario to the best of my knowledge. Outlier detection can be best done using visualizations. Box plots are the best visualizations to represent outliers in our data. Below are snapshots of box plots of each individual column of my dataset. This is called Univariate outlier detection.



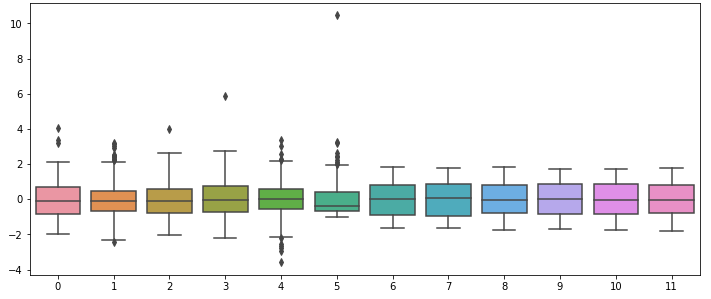
Column 1 Column 2





Column 5 Column 6

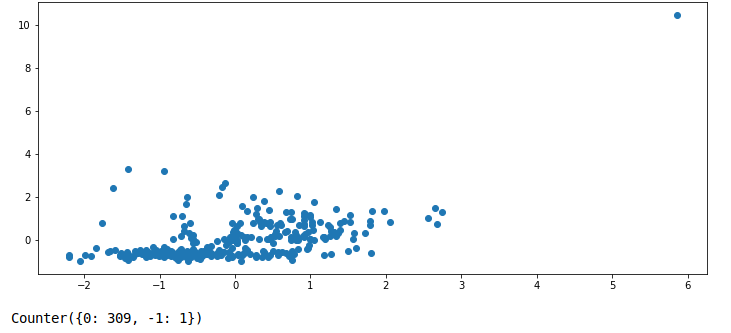
All these graphs represent outliers in the specific columns. But we are interested in the broader picture. We need to look for outliers in the entire dataset, that is, across the entire dataset, the combination of columns would have outliers. These can be detected by visualizing a box plot for all the columns. To do so, all columns should be standardized so that we can accomodate all columns in the same box plot. Below is the box plot for all columns.



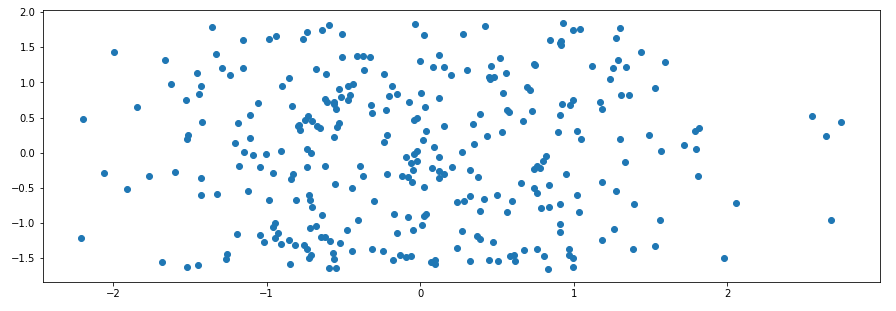
Dealing with outliers can be done in three ways:

* Deletion of the data sample : This involves deleting the data rows from the dataset. However, this method should be implemented only if the data that we are losing is minimal. If more than 15-20% of a dataset will be lost, then we should consider implementing alternate methods to deal with outliers. Based on the size of the data set, this decision should be made.
* Clamping outlier values : Outlier values can be clamped to upper and lower boundary ranges of the column. In a box plot it can be set to median+1.5\*IQR or median-1.5\*IQR.
* Setting the outlier as a missing value and imputing it : The outliers can be set as missing values and imputed using SimpleImputer()

From the above graph, we can see that Col6 (index 5) has an outlier that is very distinct from other values in the data. Also, Col4 (index 3) has an outlier whereas other columns have outlier data points very close to the whiskers. Therefore, I did not consider them to be outliers. It will be an easy task to directly remove the outliers from the data, but removing 2 rows from a data set of 310 rows only is definitely a big deal. In order to clarify if both these data points belong to the same row of data, I used DBScan multivariate outlier detection technique between Col4 and Col6 and concluded that both these data values belong to the same data sample. A scatter plot depicting the standardized values in Col4 and Col6 is represented in the below graph.



Clearly from the boxplot and the scatter plot, we can conclude that the outlier is a single data sample from the data set. Although I considered imputing the data after making these two columns as missing values, I chose not to do so as the values seem to be way out of the league. Therefor I deleted this data sample using DBScan’s counter. On deletion, the scatter plot between the two columns looks as below.



Now, we have successfully dealt with outliers in the data set.

**2.1.3 Dealing with Missing Values**

Identifying missing values can be done using the isnull() function. In our dataset, there are no missing values. Therefore, I have not implemented any techniques to exhibit dealing with missing values.

**2.1.4 Feature Selection**

Feature Selection is the process of selecting features/columns from the dataset that have the highest say in determining class labels. Few columns might have very less impact or even have no impact in determining the class label. These features might contribute in predicting the wrong class labels. Therefore, it is best to select the best features for prediction.

In the assignment, I have used the Greedy Feature selection technique to identify the columns that are most relevant. This technique not only enables ranking of features in order of their relevance but also removal of other columns that do not matter. In our data, columns 2,4,5 and 6 are ranked 1. This means that these four columns are most relevant in predicting the class label. Therefore, the rest of the code uses only these 4 columns for training and predicting classes.

In order to check if there is a difference in accuracy after implementing Feature Selection, I used the DecisionTreeClassifier and obtained the following accuracies:

|  |  |
| --- | --- |
| Before Feature Selection | Accuracy : 75.35% |
| After Feature Selection | Accuracy : 78.35% |

**2.1.5 Handling Imbalance**

Imbalance in data is a very common problem in real-world datasets. There may be more data samples for a particular class whereas some other class might have very few data samples. This may lead to an issue with the prediction of the minority class in the test data although the accuracy rates of these models maybe considerably good. In order to address this issue, we need to resample the data. The most important aspect to keep in mind here is to resample only the train data. This can be done by either:

* Under-sampling : Reducing the number of samples in the majority class
* Over-sampling : Increasing the number of samples of the minority class

In the assignment, my dataset consists of 67% majority class data samples and ~32% of minority data samples. Although this is a fairly good proportion, I chose to research in this area. Therefore, I have resampled my training data too using the SMOTE sampling technique to over-sample my data. This ensures that the minority class is represented enough in the dataset.

We use Confusion matrix to get a view of the how many of the majority and minority class data samples have been predicted accurately. In the code, this has been illustrated very well for many models with and without balancing. For instance, let us consider the RidgeClassifier() model.

Without SMOTE balancing, the confusion matrix of the entire test(validation data)looked like this:

Confusion Matrix :

[[190. 19.]

[ 25. 75.]]

Accuracy : 85.76051779935275 %

With SMOTE balancing, the confusion matrix looked like this:

Confusion Matrix :

[[155. 54.]

[ 4. 96.]]

Accuracy : 81.22977346278317 %

Although the accuracy of the model has dipped, the prediction accuracy of the minority class has significantly improved by 28%. This is the impact of sampling.

My Research topic is ‘handling imbalance’ therefore, more of this will be covered in-depth in the Research section.

**2.2 Model Selection**

The Scikit-learn package provides a variety of machine learning models to implement build our model. To do so, we need to train, validate and test our data. Since the dataset is too small, Stratified k-fold cross validation is used here to train and validate our data. In the assignment, I have chosen the following 8 baseline models to evaluate my dataset:

* RidgeClassifier()
* LinearSVC()
* GaussianNB()
* KneighborsClassifier()
* BernoulliNB()
* DecisionTreeClassifier()
* RandomForestClassifier()
* MLPClassifier()

Pseudocode of this implementation:

for every model:

for every train and test split in a 10-folded Stratified k-fold:

Fit the train\_index into the model

Predict the results for the test\_index

Calculate the confusion matrix and metrics

Balance the training data using SMOTE

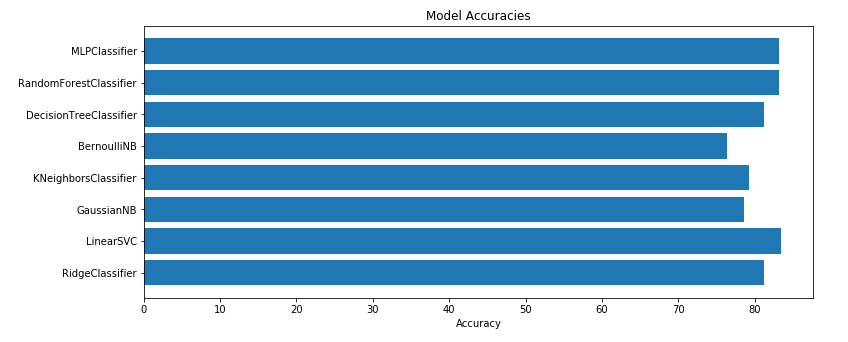
Fit the train\_index into the model

Predict the results for the test\_index

Calculate the confusion matrix and metrics

Print all metrics before and after re-sampling the data

As indicated in the pseudocode, for every model and every split in Stratified k-fold metrices are evaluated before and after re-sampling using SMOTE. The model performances are evaluated using accuracy rates and are represented in the below graph.



From the above graph, it is evident that the MLPClassifier(83.17%), RandomForestClassifier(83.17%) and the LinearSVC(83.49%) Classifier have the highest accuracies.

**1.3 Hyperparameter Tuning and Evaluation**

Every machine learning model can be further tuned in accordance with our dataset to obtain better performance metrics. This can be done by choosing the right parametes for our model. One such technique used to choose the best hyperparameters for models is the GridSearchCV method. This model takes as input the paramter grid out of which the parameters are evaluated against each other and the best combination of parameters is obtained from each model. Also, since we need to re-sample our training data, a pipeline is used with takes as input the re-sampled dataset and the classifcation model. On passing the pipeline to GridSearchCV and the fitting it to the data, the training data alone gets resampled and the model’s best hyperparameter combination is given as output.

For LinearSVC:

LinearSVC(C=1.0, class\_weight=None, dual=True, fit\_intercept=True,

intercept\_scaling=1, loss='squared\_hinge', max\_iter=1000,

multi\_class='ovr', penalty='l2', random\_state=10, tol=0.0001,

verbose=0)

Best Parameters : {'model\_\_loss': 'squared\_hinge', 'model\_\_max\_iter': 1000, 'model\_\_penalty': 'l2', 'model\_\_verbose': 0} with a score of 0.8225806451612904

For RandomForestClassifier:

RandomForestClassifier(bootstrap=True, ccp\_alpha=0.0, class\_weight=None,

criterion='gini', max\_depth=None, max\_features='auto',

max\_leaf\_nodes=None, max\_samples=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=100,

n\_jobs=None, oob\_score=False, random\_state=10, verbose=0,

warm\_start=False)

Best Parameters: {'model\_\_criterion': 'gini', 'model\_\_max\_depth': None, 'model\_\_max\_features': None, 'model\_\_min\_samples\_leaf': 2, 'model\_\_min\_samples\_split': 2, 'model\_\_n\_estimators': 50} with a score of 0.8190322580645162

For MLP Classifiers :

MLPClassifier(activation='relu', alpha=0.0001, batch\_size='auto', beta\_1=0.9,

beta\_2=0.999, early\_stopping=False, epsilon=1e-08,

hidden\_layer\_sizes=(100,), learning\_rate='constant',

learning\_rate\_init=0.001, max\_fun=15000, max\_iter=200,

momentum=0.9, n\_iter\_no\_change=10, nesterovs\_momentum=True,

power\_t=0.5, random\_state=10, shuffle=True, solver='adam',

tol=0.0001, validation\_fraction=0.1, verbose=False,

warm\_start=False)

Best Parameters : {'model\_\_activation': 'relu', 'model\_\_alpha': 0.001, 'model\_\_hidden\_layer\_sizes': (20, 2), 'model\_\_learning\_rate': 'constant', 'model\_\_max\_iter': 2000, 'model\_\_solver': 'lbfgs'} with a score of 0.8320430107526882

**3 Research – Imbalanced Data**

My research is on dealing with imbalanced datasets. I have implemented 6 balancing techniques and compared the result metrics in all these cases along with the imbalanced data. The model I have use for this is the ensemble Voting Classifier with 3 estimators. These 3 estimators are the 3 models obtained in the previous step with the best combination of hyperparameters. The 6 balancing techniques that I have implemented are:

* Random Under-Sampling
* RamdomOver-Sampling
* SMOTE
* TOMEK
* Combination of SMOTE and TOMEK
* Combination of SMOTE and ENN

I have used one view and 3 metrics to compare and evaluate these models:

* Confusion matrix
* Accuracy rate
* Specificity Score
* Geometric Mean Score

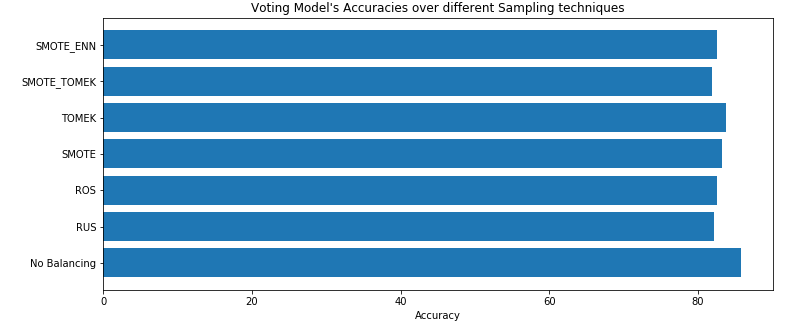
**3.1 Confusion Matrix**

Without Balancing 72 out of 100 minority samples were predicted correctly.

|  |  |
| --- | --- |
| **Sampling Method** | **Increase in Accuracy rate of Prediction of Minority Class** |
| Random Under-Sampling | 19.5% |
| Random Over-Sampling | 15.2% |
| SMOTE | 8.3% |
| TOMEK | 8.3% |
| Combination of SMOTE and TOMEK | 13.8% |
| Combination of SMOTE and ENN | 29.1% |

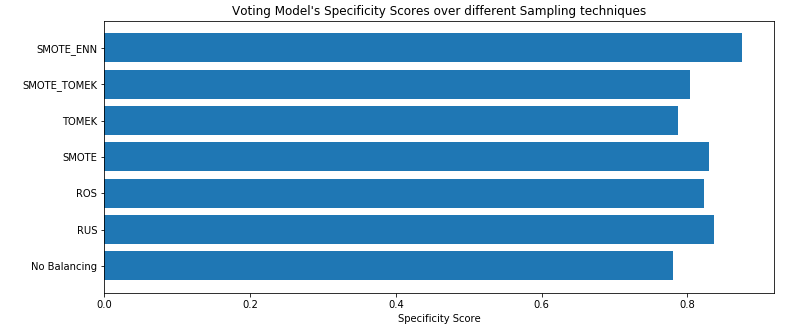
**3.2 Accuracy rate**

Re-sampling, in most cases brings down the overall accuracy rate of the model. Nevertheless, consistency of the model acroos all datapoints is more important than accuracy. So a little compromise in accuracy rate of model would do no loss. The fall in accuracy rate can be observed in the below graph after training data is re-sampled.



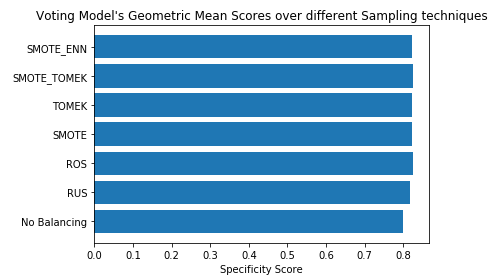
**3.3 Specificity Score**

Specificity score is the ability of the classifier to find all the positive samples. The specificity is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. [2]

It can be observed from the above graph that the specificity score of SMOTE\_ENN is the highest(>0.9).

**3.4 Geometric Mean Scores**

The geometric mean (G-mean) is the root of the product of class-wise sensitivity. This measure tries to maximize the accuracy on each of the classes while keeping these accuracies balanced. For binary classification G-mean is the squared root of the product of the sensitivity and specificity.[3]



**4 Conclusion and Future Work**

Although Scikit-learn has provided us with zillion packages to make life easier with building machine learning models, a person with no or little knowledge would still build an undrperforming model if not educated with the numerous aspects that need to be considered in building a model on a dataset. Just like practice makes a man perfect, tuning and training makes a model perfect.

On researching through sampling techniques, I realized that achieving significant leaps in certain metrics can lead to compromising other metrics. The key here is to find the right balance between these metrics. This aspect might be a great area for further research.

**References :**

[1] Dataset : <https://www.kaggle.com/sammy123/lower-back-pain-symptoms-dataset>

[2]<https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.metrics.specificity_score.html>

[3]<https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.metrics.geometric_mean_score.html>

Extensively used Lecture notes by Dr. Ted Scully to prepare this document and the code file.