

Data Mining and Machine Learning Project Report

Özgün Mutlu - Yusuf Hilooglu
s1115680 - s1111874

January 2024

1 Preliminaries

1.1 PCA

First, we wanted to apply Principal Component Analysis (PCA), but before applying that, we had to make sure if our dataset is suitable for PCA. That is why, we did a Kaiser-Meyer-Olkin (KMO) analysis. The KMO measure gave us a result of 0.81, telling us that our dataset is appropriate to apply PCA.

With the result of the KMO analysis, we decided to do PCA, if our choice of classification model benefits from it. After that, we applied PCA to our dataset. PCA reduces the dimensionality with keeping most of the variance. In our analysis, we wanted to keep at least 90 percent of the variance. With that constraint, we managed to reduce the number of dimensions from 72 to 22.

1.2 Attempts With More Basic Classifiers

Following the PCA, we tried various classification models to predict outcomes based on the transformed dataset. In this subsection, we will talk about the rather "basic" methods that we tried. Namely, logistic regression and decision trees. Since these models are often not the best with high-dimensional data, we did not expect these models to be our final ones. However, it is still good practice to try various classifiers and compare the results.

Since logistic regression draws a linear decision boundary, its performance declines drastically with high-dimensional data. Which was also the case in our analysis. After fitting to the data, logistic regression model had 83 percent accuracy.

Decision trees, in theory, can learn any non-linear decision boundary. However, this high variance also makes them prone to overfitting. That is why even though our decision tree model did not perform that poor, we decided not to choose it as our final model. Note that for some reason, our decision tree couldn't even perform much better accuracy than logistic regression. It also performed poorly with 85 percent accuracy.

In summary, we found that basic classification models perform poorly in classifying the given training data. Also, we think that if one wants to achieve lower training errors with high-dimensional data, then basic classifiers are not the correct way to go.

1.3 Advanced Classifiers

We choose two techniques to analyze in this section. Namely, multilayer perceptron classifier (MLP) and an ensemble classifier, random forest classifier.

Our MLP classifier performed really well in terms of accuracy. With tuning its hyper parameters (number of layers and their sizes) with cross-validation, we got an even higher accuracy rate, 94 percent. Which is really good in terms of training error. We also draw an ROC curve for the MLP classifier, it performed really good with an AUC of 0.98. However, since MLP is a model that has a low bias and high variance, we suspected overfitting. However, our knowledge could not go behind reducing the size and number of layers. Therefore, we wanted to go with a model that is less prone to overfitting and can actually provide a reliable estimate of generalization error.

Finally, we tried implementing the random forest classifier. Since our aim is to not only have the lowest training error, but to also give a reliable estimate of generalization error, random forest is a great with because it uses Out-of-bag error rate. Out-of-bag error tests the performance of the classifier with using the training data that it did not consider while bagging, which is approximately 27 percent of the whole data set. Therefore, it provides a good estimate of the generalization error. Therefore, we decided to do this classification task with random forests.

2 Classification Approach

2.1 Random Forests

In the previous section, we mentioned that we chose random forest classifier (RFC) as our final classifier model. How an RFC works is, it creates a certain number of uncorrelated trees that all try to classify the data. Trees are all different because in each split, attributes a tree can use to split are chosen randomly. Therefore, each tree is uncorrelated and captures a different aspect of the data. Then, when a new object is present, each tree gives a prediction and the final prediction is decided with a majority vote.

2.2 Pre-Processing

Whether or not performing PCA before RFC, or to do pre-processing in general, is a subject of research^[1]. However, since RFC does not necessarily need pre-processing, we decided not to do PCA. Same goes for removing outliers. Since decision trees are robust to outliers, we also did not have to do anything about it.

To be able to perform a validation in the end, even though oob error is a reliable estimate, we take 10 percent as the data and name it "validation set".

2.3 Tuning the Hyper Parameters

We built the model using the "RandomForestClassifier" class of the sklearn library in python. However, this class has a lot of parameters that drastically change how the model behaves. Therefore, one must do the necessary analysis to tune the parameters.

In our case, we were specifically interested in two parameters, "max_sample_size" and "max_depth". To overcome this, we used sklearn's "GridSearchCV" class with accuracy as scoring, and we also used plotting the OOB estimate with respect to a parameter.

2.3.1 GridSearchCV

What gridsearch basically does is, for the amount of possible combinations of parameters, it performs cross-validation and then returns the best parameter in terms of the chosen scoring method. When we give some potential values for max_depth and max_sample_size and ran grid search, we get that the optimal values for the parameters, with accuracy as scoring, is 25 and 4000, respectively.

2.3.2 Plotting OOB

With grid search, we are scoring in terms of accuracy. Since our goal is to perform good in imbalanced data, this scoring is not optimal. However, one cannot use OOB estimate with grid search because it does cross-validation and it is not clear what the bag is at that moment (at least that is our assumption). Therefore, to tune the parameters with respect to the oob estimate, we fit a model for each possible parameter value and analyse the graph to decide the parameters. Figure 2.1 and 2.2 represent the relation with oob estimate and the parameters. From those graphs, we decide to set "max_depth" to 15 and "max_sample_size" to 4000.

2.4 Class Imbalance

Last thing we did before building the final classifier is addressing the class imbalance issue. Since the training data is balanced and the test data is not, our idea was to adjust class weights.

Random Forest Classifier has a parameter `class_weights` that allows you to adjust the bagging distribution. Therefore, to give the minority class more focus, we set the `class_weights` as follows:

$$Weight_{c_i} = \frac{1}{prior(c_i)}, \text{ where } c_i \text{ represents the } i\text{'th class.}$$

3 Performance Estimation

In the random forest classifier, as mentioned earlier, we trust the out of bag error estimate. With our tuned parameters, the average oob estimate that we get is around (0.9, 0.93). Therefore, if we're being pessimistic, the generalization error of our classifier is $(1 - oob) \cdot 100$. In average, the estimated test performance is 8 percent.

We also did validation with the validation set that we split from the training data. The accuracy on the validation set was also 8 percent. Since we get 8% from both our estimation methods, we are confident in it.

Finally, we drew an ROC curve on the validation set to see how well our classifier works. The result is shown in Figure 3.3. (Overleaf wanted it to be 3.3 instead of 3.1)

The estimated error rate is 8%

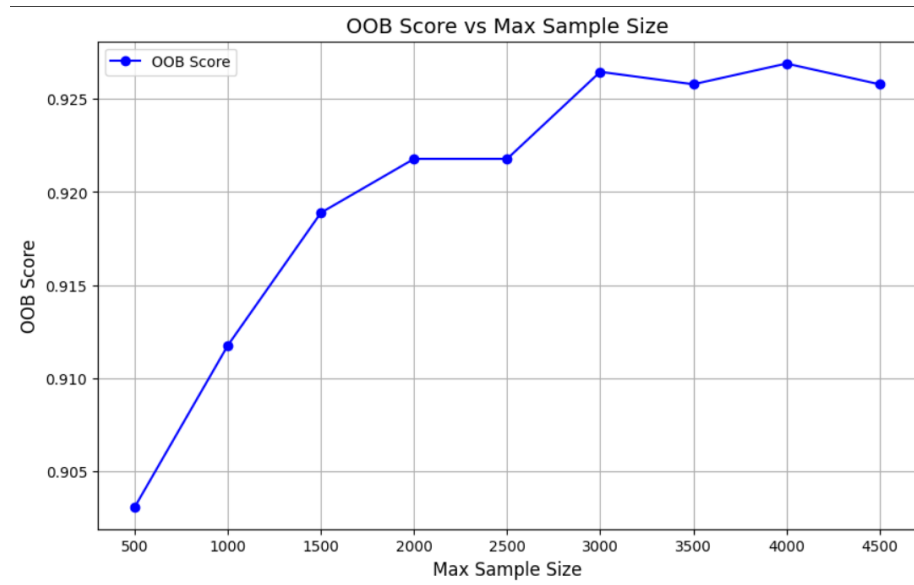


Figure 2.1: OOB Score - Max Sample Size

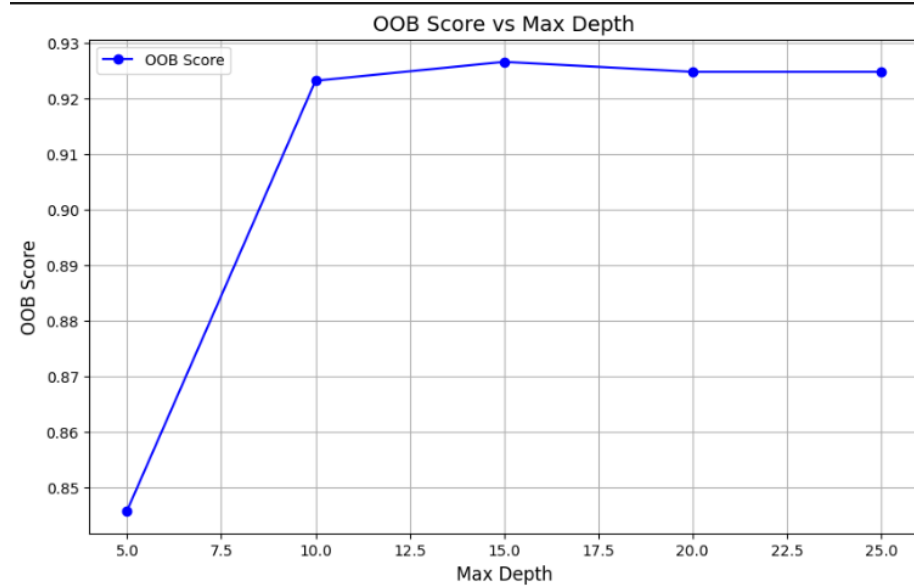


Figure 2.2: OOB Score - Max Depth

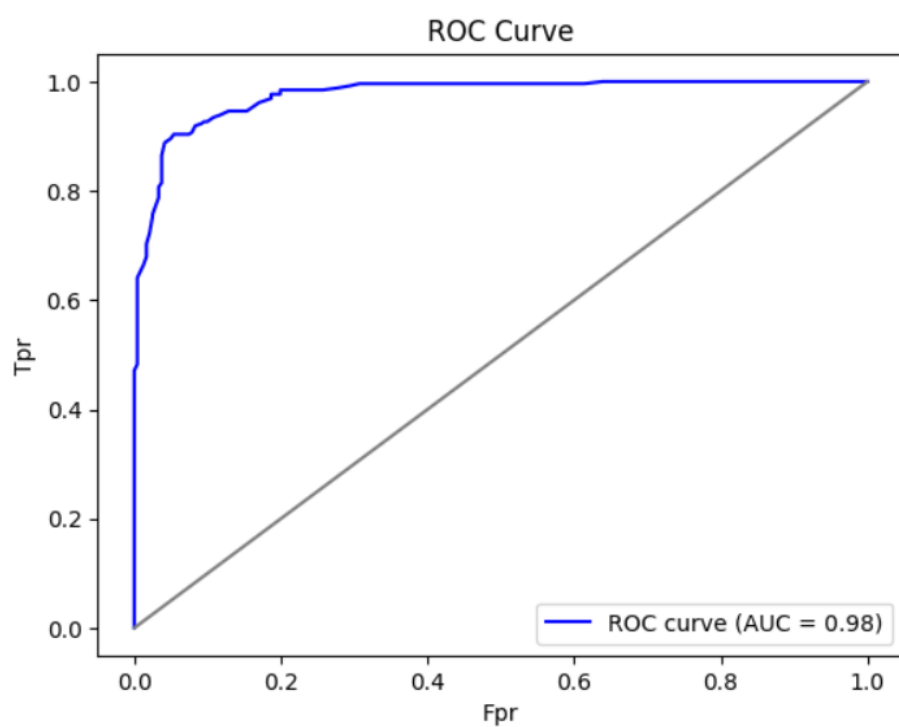


Figure 3.3: ROC Curve on the Validation Set

4 Other Matters

We named the first class, "0" and the second class "1". Since there were no regulations on what their names should've been, we hope that this is an acceptable naming.