Attached file: hw2.ipynb

I open the data file called pd_speech_features.csv in the pandas back dataframe format. The data is 756 rows with 755 columns, and from these columns, I take the class column as the target variable, which has a value of 0 or 1. The problem is a two-class classification. From the rest of the columns, I leave the id column because it is only a sample code from which the data is collected and has nothing to do with the problem space. I take these 753 columns as the set of dependent variables. All columns have a numeric value. Using sklearn.model_selection, the train_test_split function shuffles the data and separates the test and training data with a ratio of 30 to 70. In order to get better results from the classification and performance of the models, I standardize the data before building the models. The StandardScaler function of sklearn.preprocessing performs standardization on both training and test sets using the mean and variance of the training set.

Decission Tree

The first model built on the clf6 data is a decision tree with information gain criteria, which builds the tree in the direction of the best branching to reduce entropy. This model is built with sklearn functions. It has a depth of 8. It is almost completely fitted on the training data and has an accuracy of 97. Each leaf contains at least 1 piece of data, and each node has at least 2 pieces of data for branching, and tree pruning criteria are set to a minimum. The first attempt to build a tree for this problem was a tree with a depth of 10, which had a result of nearly 70 on the test, and to reduce the Orfit error, I set a limit of 8 for the depth, and in the final model, I classify the test data with clf6. The following results are obtained respectively on the test set and after the training:

```
              precision    recall  f1-score   support

           0       1.00      0.89      0.94       123
           1       0.97      1.00      0.98       406

    accuracy                           0.97       529
   macro avg       0.98      0.94      0.96       529
weighted avg       0.97      0.97      0.97       529

              precision    recall  f1-score   support

           0       0.71      0.51      0.59        69
           1       0.81      0.91      0.86       158

    accuracy                           0.79       227
   macro avg       0.76      0.71      0.73       227
weighted avg       0.78      0.79      0.78       227

[[ 35  34]
 [ 14 144]]
```

The main weakness of the Marbouz model is the detection of class zero. Modeling was done several times and each time the accuracy was around 80%. The model does not perform badly.

Random Forest

The second model built on the clf7 data is a random forest in which there are 100 trees with information gain criteria as categories, and each one is built by bootstrap sampling on the data and on the features for the best branching to reduce entropy. have became. Tree construction is done using sklearn. Education classification done with 100% accuracy. By evaluating the test data label with clf7, the following results are obtained.

```
              precision    recall  f1-score   support

           0       0.87      0.49      0.63        69
           1       0.81      0.97      0.88       158

    accuracy                           0.82       227
   macro avg       0.84      0.73      0.76       227
weighted avg       0.83      0.82      0.81       227

[[ 34  35]
 [  5 153]]
```

Compared to the decision tree, the result is better. The difference in accuracy is small and the most change is found in the detection of zero class and the improvement of the model is related to this part of precision.

XGBoost

The third model or clf8 is an XGBoost model built with the xgboost library. Default parameters are used for this model, the number of trees is 100, the learning rate is 0.300000012, and the maximum depth of each tree is 6. Each time, each tree is built iteratively from the previous tree so as to improve the previous tree. The model is fitted on the training data without error. The test results are as follows:

	precision	recall	f1-score	support
0	0.95	0.59	0.73	69
1	0.85	0.99	0.91	158
accuracy			0.87	227
macro avg	0.90	0.79	0.82	227
weighted avg	0.88	0.87	0.86	227

```
[[ 41  28]
 [  2 156]]
```

As expected, clf8 is in a better condition than both previous models. This model has an accuracy of 87 and clearly performs better in class zero precision. This improved model is the same random forest as it has its positive features, including that it uses the consensus vote of several categories that are randomly built on samples of features and data, and it has this advantage. which includes only trees that complete each other in the constructed samples.

SVM

The clf9 model is a support vector machine. With the regularization factor value of 5, radial basis function is used as kernel to transfer the data to the new space. The training data are classified with full accuracy and the results of the model on the test set are as follows:

	precision	recall	f1-score	support
0	0.90	0.52	0.66	69
1	0.82	0.97	0.89	158
accuracy			0.84	227
macro avg	0.86	0.75	0.78	227
weighted avg	0.85	0.84	0.82	227

```
[[ 36  33]
 [  4 154]]
```

svm performance is lower than xgboost and it is random in janga size. That is, it is better than the clf6 model, but due to the closeness of the numbers, the best way to compare these 3 models is the average k-fold cross validation method, but because the clf8 model, which is clearly better than the other two in the first level, we have this comparison as a result to find It does not have the best model.

MLP

The first clf1 model is a multilayer perceptron network built with sklearn.neural_network.MLPClassifier. As the activation function, I used the non-linear sigmoid or logistic function. The network has two layers and the number of nodes in the hidden layer of the network is 20, and a maximum of 1000 ipacks is spent for training. The learning rate is equal to the pre-loan amount of 0.001 taken. After training, this model is fully fitted on the training data, and after classifying the test data and measuring the error values, the performance of the model is as follows:

	precision	recall	f1-score	support
0	0.68	0.71	0.70	69
1	0.87	0.85	0.86	158
accuracy			0.81	227
macro avg	0.78	0.78	0.78	227
weighted avg	0.81	0.81	0.81	227

Also, the confusion matrix for the test data is as follows:

```
[[ 49  20]
 [ 23 135]]
```

The performance of clf1 model is better in the detection of the second class (class 1) according to all criteria of f1, recall and precision, and the overall performance is 81%. Considering the difference in accuracy on the test and training sets, I guessed that it has a little overfit error. . I built the same model with more different parameters and through several trials and errors, I trained models with less IPAC or increased and decreased the number of nodes in the middle layer. But the best results obtained were around 80 to 85 percent, and there was no improvement in the test results. So I reported the same as the result of the first model. In terms of accuracy, this model is on par with clf6, clf7, and clf9 models, but it performs much better than them in terms of recall value in zero class.

ELM

The second model is again a neural network, this time trained using the ELM method. I wrote and implemented the construction and training code myself in different functions. The ELM

function is a network building and training function that first determines the number of features and inputs, then takes random numbers for the vector of weights between the inputs with the hidden layer. It determines the bias weights randomly and finally, from the solution of the matrix equation and the calculation of the inverse matrix, it finds the weights of the middle layer to the output so that the output vector is equal to the mean of the target variables. It uses the hidden_nodes function to calculate the value in the nodes of the hidden layer. This function applies a mathematical function as an activation function on the values before sending the values to the next layer (output). I used sigmoid function here which is defined in sigmoid function. Finally, predict evaluates and classifies the data on the input and output sets. I made the model of this part with 35 nodes in the hidden layer and in the form of two edges. The results of the training and its confusion matrix are as follows:

	precision	recall	f1-score	support
0	0.62	0.45	0.52	123
1	0.85	0.92	0.88	406
accuracy			0.81	529
macro avg	0.74	0.68	0.70	529
weighted avg	0.79	0.81	0.80	529

```
[[ 55  68]
 [ 33 373]]
```

And after the classification of x_test data, the accuracy value of the model is as follows:

	precision	recall	f1-score	support
0	0.69	0.45	0.54	69
1	0.79	0.91	0.85	158
accuracy			0.77	227
macro avg	0.74	0.68	0.70	227
weighted avg	0.76	0.77	0.75	227

```
[[ 31  38]
 [ 14 144]]
```

The accuracy of clf2 is low and even by changing the parameters on the training data, it does not fit completely or even with results that are significantly better than this. Of course, due to the randomness of half of the weights of the model, different results can be obtained with fixed parameters, but even in the best case, it still performs weaker than clf1. From the results of the table above, clf2 has a better performance on class 1 and even this performance on recall is better than clf1. It means that it is better in confidence in label 1, but it has acceptable accuracy on zero class, but on the other hand, recall, or in other words, confidence in zero label is worse

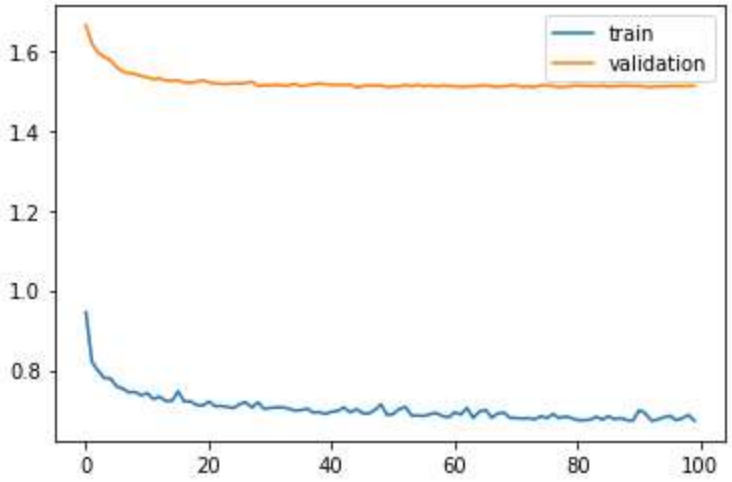
than 50-50 chance! And this value also affects f1 and the whole model cannot be used or relied on. It should be noted that in clf6, the accuracy was only slightly better than this, but in terms of the other criteria mentioned, the decision tree performs much better than clf2.

AutoEncoder

For this part, I encoded and decoded the data using a neural network. All these operations, including the definition of the network and the architecture of the layers, and training using tensorflow.keras functions and classes. In each step, I used a two-layer network, and in the first time, I reduced the data to ¼, and in the network bottleneck section, the number of features was reduced to 1/8 of the initial value. In the decoder section, I first multiplied the data by 8 and then by 4 in order to restore the original values. I used sigid as output activator function. Then I trained the model with the same x_train and in 100 ipak with mse as the error pan and used the x_test set to calculate the error. All these sections are performed without an observer, that is, without the presence of class target variables, so the encoder data and results can be considered as a separate problem, and the use of test data to evaluate the encoder does not affect the results of the classifiers in this section. The constructed network is as follows:

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 753)]             0
_________________________________________________________________
dense (Dense)                (None, 753)               567762
_________________________________________________________________
batch_normalization (BatchNo (None, 753)               3012
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 753)               0
_________________________________________________________________
dense_1 (Dense)              (None, 188)               141752
_________________________________________________________________
batch_normalization_1 (Batch (None, 188)               752
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 188)               0
_________________________________________________________________
dense_2 (Dense)              (None, 94)                17766
=================================================================
Total params: 731,044
Trainable params: 729,162
Non-trainable params: 1,882
```

Now I used the encoder part of this network to reduce the number of independent variables of the classifier as a compressor. This encoder increases the number of features regardless of their class in the original problem from 753 to 94 features. The diagram of the training process of the autoencoder network is as follows:



Now, on this x_train_encode compressed dataset, I built and trained three clf3, clf4, and clf5 models. And then I compressed the test data with the same model and did the classification on x_test_encode to calculate the accuracy.

The first model of this part or clf3 is a perceptron. I built and trained this simple model with sklearn.linear_model.Perceptron. The results of the education classification are as follows:

```
              precision    recall  f1-score   support

           0       0.64      0.82      0.72       123
           1       0.94      0.86      0.90       406

    accuracy                           0.85       529
   macro avg       0.79      0.84      0.81       529
weighted avg       0.87      0.85      0.86       529

[[101  22]
 [ 56 350]]
```

And by classifying the test data with clf3, the result of the model's performance is obtained as follows:

```
              precision    recall  f1-score   support

           0       0.70      0.68      0.69        69
           1       0.86      0.87      0.87       158

    accuracy                           0.81       227
   macro avg       0.78      0.78      0.78       227
weighted avg       0.81      0.81      0.81       227

[[ 47  22]
 [ 20 138]]
```

On the test data, the result of the linear perceptron classifier for compressed data could not be better than this value between 75 and 82, and in this implementation, the accuracy of 81 was obtained. This data cannot be linearly separated with a hyperplane. An obvious difference between this model and clf2 is the improvement of precision and total recall. clf3 performs better than clf2. The model is still better overall in class 1, but it has more uniform accuracy on two classes than clf2.

The second model of this section, clf4, is the same multilayer perceptron network that I used for clf1, and this time it works with compressed data as input (the architecture and training method are exactly the same, only the number of inputs, which is the number of data features, has been reduced) . The model fits perfectly on the training data and gives 100 accuracy. I'm sorting on the x_test_encode collection. The accuracy of clf4 is as follows:

```
              precision    recall  f1-score   support

           0       0.77      0.62      0.69        69
           1       0.85      0.92      0.88       158

    accuracy                           0.83       227
   macro avg       0.81      0.77      0.78       227
weighted avg       0.82      0.83      0.82       227

[[ 43  26]
 [ 13 145]]
```

In all reported parameters, the performance of clf4 and clf1 is similar to each other. In some cases it is a little better and in others it has decreased. My purpose of using this model was to see the effect of compressing the data on the first model, and since the criteria are very close, I guess that compression has no effect on the classification result of this model, but the final report and review of this issue will be related to the results. I leave the results obtained from the average of k fold cross validation in the next section.

The third model of this section, clf5, is a network that is trained with the ELM algorithm on the compressed data of the x_train_encode set. The description of clf5 model construction is the same as for clf2. The results of the training are as follows:

```
              precision    recall  f1-score   support

           0       0.77      0.48      0.59       123
           1       0.86      0.96      0.90       406

    accuracy                           0.84       529
   macro avg       0.81      0.72      0.75       529
weighted avg       0.84      0.84      0.83       529

[[ 59  64]
 [ 18 388]]
```

And the result of the model's performance on the test:

```
              precision    recall  f1-score   support

           0       0.60      0.41      0.48        69
           1       0.77      0.88      0.82       158

    accuracy                           0.74       227
   macro avg       0.68      0.64      0.65       227
weighted avg       0.72      0.74      0.72       227

[[ 28  41]
 [ 19 139]]
```

The decline in model quality is evident in all sections and all criteria. If we compare clf2 with clf5, data compression for this problem on the ELM model not only does not improve, but also makes the model more inefficient than before in all evaluation cases.

5 Fold Cross Validation

So far, I made 9 models, 3 of which used the autoencoder compressed data. Among these 9 models, 3 to clf6, clf5 and clf2, which were weaker than the others, and clf8 showed better results than the others, but it is not easy to choose between the others and which model is better based on these results, because the values obtained are close were also In this part, I made all the models again this time with all the training data at once and with the cross validation method and cv = 5 and averaged the accuracy criteria on each one. For 3 models that worked with encoded data, compression was also done with the encoder of the previous section. Standardization is applied to all data. It means that standardization is done before the validation set is separated. This may affect the accuracy because in all pre-processing training and testing should be done separately, but here I ignored this effect. With accuracy, the best model that was obtained was clf7, which was made by random forest method. Of course, clf9 and clf8, both of which had close to this. The results are in the table in the next part. For this model, the confusion matrix values:

```
[[102  90]
 [ 35 529]]
```

Compare the results with the article

A comparative analysis of speech signal processing algorithms for Parkinson's disease classification and the use of the tunable Q-factor wavelet transform

https://doi.org/10.1016/j.asoc.2018.10.022

In this article, the goal is to predict and diagnose Parkinson's disease by examining the audio signals that were worked on in this exercise. In this work, the goal is to select a series of features from the data as a baseline feature and then select another series of features with feature selection methods. The selected collections are tested in different classification models. In the best model obtained in this research, a set of 50 features was selected and listed in the last row of the table below. In this article, only f1, accuracy and MCC criteria are reported and reviewed. The results of this model are better than all the models I obtained in this report. I think the main reason is that there are some of these features that do not make a significant difference in the diagnosis of Parkinson's disease. And the purpose of this article is to identify the features of the processed audio tapes of the samples that are related to the disease. And if the assumption is correct that some of these columns have no effect on the classification result, then their presence in the models will reduce the performance of my neural network models. The second reason can be pre-processing. I did nothing to scale or transform the data other than standardizing. Making these kinds of changes or using dimensionality reduction strategies may give better results for my network models. But for xgboost, random forest, and svm, the performance is almost the same as the selected model of the article, with the difference that the article reached this performance with 50 features using a suitable feature selection strategy, and I used all the data columns and a bulky model to I got this answer.

Method	Accuracy	Precision	Recall	f-measure
MLP	0.75	0.78	0.75	0.76
ELM	0.78	0.77	0.78	0.77
AutoEncoder with Perceptron classifier	0.77	0.78	0.77	0.77
AutoEncoder with MLP classifier	0.80	0.80	0.80	0.80
AutoEncoder with ELM classifier	0.80	0.78	0.80	0.79
Decision tree	0.78	0.77	0.78	0.77
Random Forest	**0.85**	**0.84**	**0.85**	**0.83**
XGBoost	0.83	0.83	0.83	0.82
SVM	0.82	0.81	0.82	0.81
	0.86	Not reported	Not reported	0.84