BLG 454E Learning From Data (2019)

Term Project Report

Mareva Zenelaj Alperen Ugus Orhan Kurto

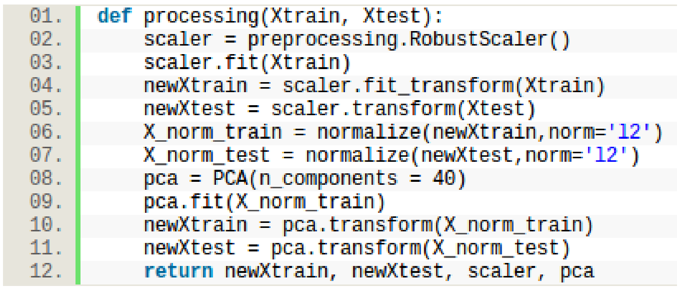
***Abstract*—Classifying autistic and not autistic people based on the Euclidian distance of the shapes of the brain. We trained a neural network and support vector machines with principle component analysis to process and train the data.**

I. Introduction

In this project we were ought to model and train a classifier that would discriminate between autistic and non-autistic people by using the difference between the shape of the regions as features. Firstly, we used a few fully connected layers with a Softmax or sigmoid activation layer at the end, yet it did not perform successfully which lead to switching to SVM with PCA. After data preprocessing and feature extraction, we fine-tuned the SVM model's parameters using cross validation. The most recent score is 60% accuracy and the Kaggle rank is 53 under the team name 150150906_150130904_150160158.

II. DATA SET USED

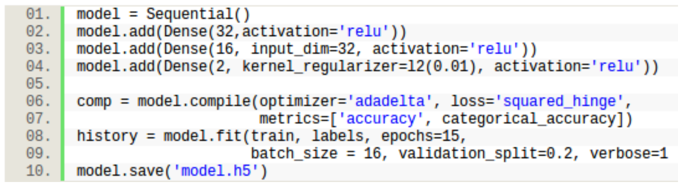
Regarding data preprocessing, the first thing we worked on was scaling the data and since sklearn has a ready built-in function for the latter called RobustScaler, we used that scaling function since it represented the data better than other ones. After scaling the data, we normalized it by using the l2 norm and following that we used PCA to extract features and reduce the dimensionality of the data. While trying several values about the number of components, the best result was given if 40 is used. Other options were: 32, 64, and 128, 64 worked nearly as good as 40, while the others performed badly. In this step, as shown in the code attached, we also returned the Scaler and PCA models in case, new test file is inputted.



```
def processing(Xtrain, Xtest):
    scaler = preprocessing.RobustScaler()
    scaler.fit(Xtrain)
    newXtrain = scaler.fit_transform(Xtrain)
    newXtest = scaler.transform(Xtest)
    X_norm_train = normalize(newXtrain,norm='l2')
    X_norm_test = normalize(newXtest,norm='l2')
    pca = PCA(n_components = 40)
    pca.fit(X_norm_train)
    newXtrain = pca.transform(X_norm_train)
    newXtest = pca.transform(X_norm_test)
    return newXtrain, newXtest, scaler, pca
```

III. METHODS USED

Firstly, we decided to use a neural network model which consisted of 2-4 fully connected layers and a Softmax layer at the end to decide the category of the classification. We decided to tune these parameters by manually changing and evaluating the parameters. We changed the number of epochs, the batch size, the validation split and the dropout value in the layers if used. The final parameters were 15 epochs, 16 batch size, 0.15 validation split and a 0.15 dropout value. This resulted in 59% training accuracy and 56% validation accuracy, while the accuracy of the test set was 50%. However, we tried to also change the optimizer, the loss function and even the accuracy metrics as indicated in the code shown below, and even though the specifics given in image1 resulted in a validation accuracy of 66.67%, it did not perform well on the test set.

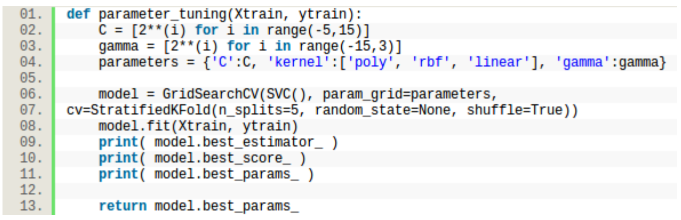


```
model = Sequential()
model.add(Dense(32,activation='relu'))
model.add(Dense(16, input_dim=32, activation='relu'))
model.add(Dense(2, kernel_regularizer=l2(0.01), activation='relu'))

comp = model.compile(optimizer='adadelta', loss='squared_hinge',
                     metrics=['accuracy', categorical_accuracy])
history = model.fit(train, labels, epochs=15,
                    batch_size = 16, validation_split=0.2, verbose=1
model.save('model.h5')
```

Since the first model we tried resulted in unsatisfactory outcomes, we decided to use SVM and PCA so that the dataset is more compact and the features are reduced to the ones that are distinguishable. For this task we decided to use sklearn and not Keras since sklearn offers many utilities such as SVM, Cross Validation, Grid Search and so on. Firstly, after we decided to keep only 40 features, we tried a simple SVM model with gamma set to automatic and C to 100 using radial basis function (RBF) and it resulted in 55% testing accuracy.

After several tries of manually changing the parameters of the support vector machine model, we decided to tune the parameters by using a grid search which would parse through given gamma and C values, use cross validation to train and test the model and finally output the best model and parameters possible in the given ranges. Besides grid search, we also tried a randomized grid search which performed slightly worse than the grid search, subsequently we decided to keep using grid search. The results and the full description of the parameters is given below. Grid search also uses cross validation as a technique to learn the best estimator. We set the folds parameter to 5 and10, yet 5 seemed to work better.

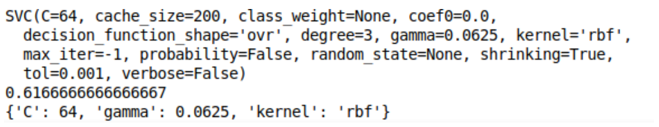


```
def parameter_tuning(Xtrain, ytrain):
    C = [2**(i) for i in range(-5,15)]
    gamma = [2**(i) for i in range(-15,3)]
    parameters = {'C':C, 'kernel':['poly', 'rbf', 'linear'], 'gamma':gamma}

    model = GridSearchCV(SVC(), param_grid=parameters,
cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=True))
    model.fit(Xtrain, ytrain)
    print( model.best_estimator_ )
    print( model.best_score_ )
    print( model.best_params_ )

    return model.best_params_
```

Regarding the gamma and C range, as stated in [1], it is recommended that a grid search on C and gamma results successful if exponentially growing sequences are used such as = [2^{-5},2^{-3}…2^{15}]. We set the range of gamma from -5 to 15 and the range of C from -15 to 3.



```
SVC(C=64, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.0625, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
0.6166666666666667
{'C': 64, 'gamma': 0.0625, 'kernel': 'rbf'}
```

As shown we have broadly used sklearn, the library python offers and Keras at the same time. We trained the models while using Jupyter Notebook and shared the code among us with Github. The main model used is SVM with different parameters.

IV. RESULTS

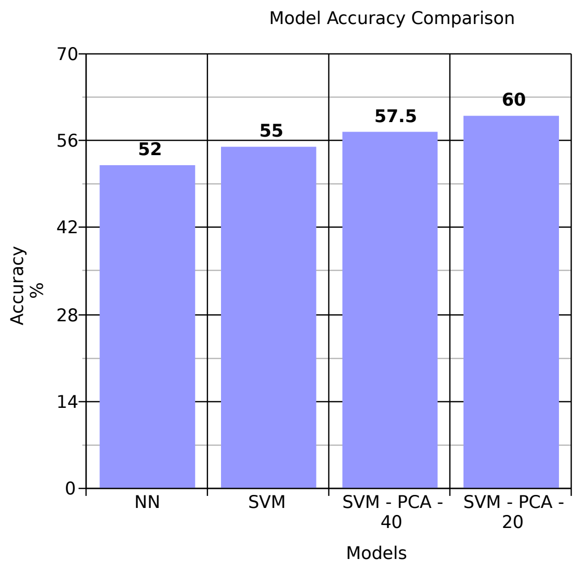
The final submission on Kaggle resulted with an accuracy of 60% and since we had tuned the parameters and processed the data already, we didn't repeat any of those steps anymore. Our Kaggle team name is 150150906_150130904_150160158 ranked 53rd with an accuracy of 60%. The model trains for about 0.016 seconds and gives the mentioned result. The submission file that reached 60% was posted as a late submission.

The graph shown in the conclusion section depicts the main models we have used in the timespan of this project. The highest result, as expected, is achieved when using SVM with PCA.

V. CONCLUSIONS

The results is quite low, given that the dataset is small, even though there are a lot of features, given that the sample size is just 120, the proportions do not promise a good model. Even after we used PCA to extract features, 40 features and 120 samples would again not result in a high accuracy.

Besides the features-sample size ratio, the dataset itself is difficult to train, given the complexity of the problem. If a person is autistic or not does not rely only on the differences of the shapes of the brain regions. We believe the mentioned reasons are the key reasons why the model did not perform as well as we expected.



VI. REFERENCES

[1] C. Hsu, C. Lin and C. Chang, "A Practical Guide to Support Vector Classification", *Csie.ntu.edu.tw*, 2016. [Online]. Available: https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf . [Accessed: 30- May- 2019].