Student Number: 229539 & Kaggle Team Name: Dimitris Papasotiriou

1. Approach

The first assumption was to train a selection of models, in the training dataset, using hyperparameter optimization for each model. Then, after having the results of all the models, make a prediction to the testing dataset, given by the Kaggle competition, by using the model which returned the best result to the aforementioned process. Hyperparameter optimization is a technique widely used which objectively search different values for model hyperparameters and then choose a subset that results in a model that achieves the best performance on a given dataset. A wide range of models and hyperparameters could be tested during the investigation, however, due to the amount of computation power and time required to train each model, the selection had to be trimmed.

My approach was then limited to one of the most robust models used for binary classification tasks, referring to Supported Vector Machines (SVM). In general, the main objective of the (SVM) is to find the optimal hyperplane to distinguish two classes in each dataset. Specifically, the optimal hyperplane is the one which has the biggest margin because it not only classifies the existing dataset but also helps predict the class of the unseen data. The inspiration of choosing the (SVM) classifier as the solution to the competition arose after having a comprehensive reading in what the authors proposed in [1] and [2] where they combined deep learning methods, such as CNN's with (SVM) classifier for image classification. More precisely, instead of using a typical fully connected layer with a SoftMax activation function, they chose to use a linear (SVM) classifier, achieving remarkable results. In addition, (SVM) works effectively for high-dimensional datasets (i.e., like the one we deal with), since the complexity of the training dataset in (SVM) is generally characterized by the number of support vectors rather than the dimensionality.

The main hyperparameters of an (SVM) are three including: the kernel function, the C penalty, and the gamma value. Kernel function refers to the type of the data transformation that the classifier will do to be able to separate the classes. Specifically, is not always possible to use lines or planes (e.g., non-linear example), thus (SVM) using a technique known as the kernel trick, maps the data to a different space where a linear hyperplane can be used to separate classes. The forms it can take are the following: linear, poly, rbf, sigmoid, and precomputed. C penalty is similar to the regularization term, as it controls the error rate. Whereas, gamma value defines how far the influence of a single training example reaches.

The problem was then approached by experimenting with various values of the (SVM) hyperparameters. The hyperparameters that were tested for each model during the investigation were generic (e.g., scaling methods or feature selection methods) as well as specific (e.g., size of margin separated hyperplane SVM). First, RBF kernel was used, as it has been shown remarkable results in image classification tasks. Next, to avoid the overfitting, especially in a domain adaption problem we deal with, very small log scaled values of C lower than 1 were tested. Doing that, we increase the error rate by allowing the classifier to accept more misclassified labels. Then, similar to the C values, a log scaled selection of small values was also investigated for the gamma parameter. Hence, we get a more linear curve, as even the far away points get considerable weight. Finally, the general hyperparameters was referring to a selection of three different feature scaling methods and one feature selection method.

2.Methodology

To begin with, the model was trained on both complete and incomplete training data. Training the classifier with more examples will help it to avoid any type of spurious correlations that may occur, thus preventing it from overfitting. For instance, the classifier might learn to associate "seas" with "exciting-place-to-be", which is not true in general. Next, all the missing data in the additional training was handled by using the technique of Imputation. More precisely, the data was split by class with the mean value for each feature calculated. Afterward, all the missing values were filled with the mean of that value for that class. This specific technique was chosen because, according to the authors in [3], it has been proved that it improves the classifier's results. Normally, the same Imputation technique would have to be applied in the testing dataset, however, there were not missing values found.

Following the completion of the missing values, the proportion of the two classes, in the dataset, was inspected for its balance. It was noticed that the training dataset was nearly balanced but did not agree with the label proportions of the testing dataset. Specifically, the major class of one dataset was the minority class of the other. In general, the main problem that occurs when having a non-fully balanced dataset is that there are too few examples, of the minority class, for a model to effectively learn the decision boundary. Hence, the classifier would struggle to classify the minority group and would fail miserably on a dataset with a large proportion of this group. The problem then was tackled by using the technique of SMOTE oversampling [4]. More precisely, the technique is working by firstly selecting an instance "a" from the minority class (i.e., in our case "exciting-place-to-be") and then find its k-nearest minority class neighbors. Then, the synthetic instance is created by randomly selecting one of the k-neighbors (e.g., "b") and connecting them to form a line segment in the feature space. Finally, the synthetic instances are generated as a convex combination of the two chosen instances "a" and "b".

Another pre-processing task that needed to be accounted was the scaling of the dataset features. Considering that the training dataset is a concatenation of features that have been extracted from different sources, we recognize that the range of the values will differ. Feature scaling is crucial for the SVM classifier which consider distances between observations, as the distance between two observations differs for non-scaled and scaled cases. As we have already stated, the decision boundary maximizes the distance to the nearest data points from different classes. Hence, the distance between data points affects the decision boundary SVM chooses. In other words, training an SVM over the scaled and non-scaled data leads to the generation of different models. In comparison to the aforementioned data preparation related tasks, the model's performance was tested for more than one feature scaling method. More specific, two types of Standardization including the: StandardScaler [5] and MaxAbsScaler [6], and one type of Normalization including the: Normalizer (using L2 norm) [7], provided by sci-kit library, were investigated during the hyperparameter optimization later.

The idea behind StandardScaler is that it will transform the data such that its distribution will have a mean value 0 and standard deviation of 1. On the other hand, MaxAbsScaler which is recommended for sparse datasets (e.g., the training dataset suffers from a lot of zero data points, caused by the ReLu activation function that was applied during the AlexNet convolutional layers), scales and translates each feature individually such that the maximal absolute value of each feature in the training set will be 1.0. As for the Normalizer, it works by rescaling the values into a range of [0,1]. The Normalizer has the benefit of affecting the dataset's outliers since, unlike the other two methods, it has a bounding range.

The final pre-processing task that was applied to the dataset, refers to the feature selection. As previously mentioned, the dataset includes two types of features, (i.e., CNNs features and GIST features), 4608 in total, which we do not know if they are equally important. Thus, it is desirable to reduce the number of input variables to both reduce the computational cost of the algorithm and, in some cases (e.g., if the right subset is chosen), to improve the performance of the classifier. Since the training dataset includes labels, a filter-based feature selection method, which uses statistical measures to score the correlation between input variables that can be filtered to choose the most relevant features, was used.

More precisely, SelectKbest [8] method provided by sci-kit module was used, which simply retains the first k features of the training dataset with the highest scores according to the score function that is given. The score function that was given to the method was the f_classif [9], which computes the one-way ANOVA F-value (i.e., evaluate the ratio of two variances) between the class labels and the features when having classification tasks.
The main reason a filter-based feature selection method was used is that, unlikely to other methods (e.g., wrapper methods), they do not involve training the model, thus is less computationally expensive, hence easier to test.

After the completion of all the data preparation tasks, the SVM classifier is ready to be fit into the training dataset and then predict the unlabelled testing dataset, that is given for the competition. However, to form an opinion of how the classifier would perform on unseen data, stratified k-fold cross-validation [10] procedure took place. More precisely, this technique is working by using a limited sample to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. Also, unlikely to the other related techniques, it generates test sets such that all contain the same distribution of classes, or as close as possible, thus each sample will be balanced similar to the entire training dataset. Since the incomplete

training data was added to the training dataset, there was enough data to be possible of using k equal to 5. This number of folds should be good enough, while 5 and 10 are values that have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance [11].

However, using the stratified k-fold cross-validation procedure, an important factor that needs to be considered is that any type of transformation must be only applied to the training dataset. This is because any type of data leakage, where data preparation techniques such as feature selection or scaling input values are applied to the entire dataset, needs to be prevented. This is data leakage since it shares knowledge of the test dataset (such as observations that contribute to mean or maximum known value) with the training dataset, and in turn, may result in overly optimistic model performance. To reassure that this is not the case in our testing, a Pipeline provided again by the sci-kit module [12], was created. Pipeline ensures that the sequence of operations (i.e., in our case oversampling, scaling, feature-selection, SVM fit), is defined once and is consistent when used for model evaluation or making predictions. In other words, any type of transformation during the cross-validation will only be applied in the training data, hence we avoid any possible data leakage.

As already been stated, the model's performance was evaluated for various hyperparameters, either general or specific, to obtain its best possible score. This was achieved by creating a Grid Search using GridSearchCV [13], which defines a search space as a grid of hyperparameter values and evaluates every position in the grid. Specifically, Grid Search was preferred over other search spaces (e.g., Random Search) since it requires less time to execute and we also have an intuition about which combinations are known to perform well generally. The technique had then allowed me to perform an investigation including the hyperparameter tuning as it evaluates models for a given hyperparameter vector using stratified k fold cross-validation (when the input model is a classifier and not a regressor). More precisely, it requires two arguments including the model, which is optimized (e.g., in our case the pipeline that was created was used, for the reason we previously mentioned) and the search space, which was defined as a dictionary where the names are the hyperparameter arguments to the model and the values are discrete values.

The metric that was used for the model's evaluation during the hyperparameter optimization was ROC AUC score [14], where it computes the area under the ROC curve, which summarizes the trade-off between the true positive rate and false-positive rate for a predictive model using different probability thresholds. This metric was preferred since it is appropriate when the observations are balanced between each class, as in our case, and it has been shown that is often preferred over other metrics for binary classification.

Finally, after the investigation was finished, the most important attributes which include the best score observed and the hyperparameters that achieved the best score, were obtained. Next, the entire training dataset was pre-processed using the optimum general hyperparameters, while the testing dataset was transformed based on that fit. Then, the classifier was fit in the training dataset, and eventually was used to predict the unlabelled testing dataset.

3. Results

In this section, the results of the model's performance on the validation data, during the hyperparameter optimization, are reported. Specifically, the results have been separated into three figures, based on the scaling method that was used in the features. All the figures illustrate the hyperparameter search against the average score of the model, where gamma represented in the x-axis whereas, C penalty is represented as a different curve (colour line).

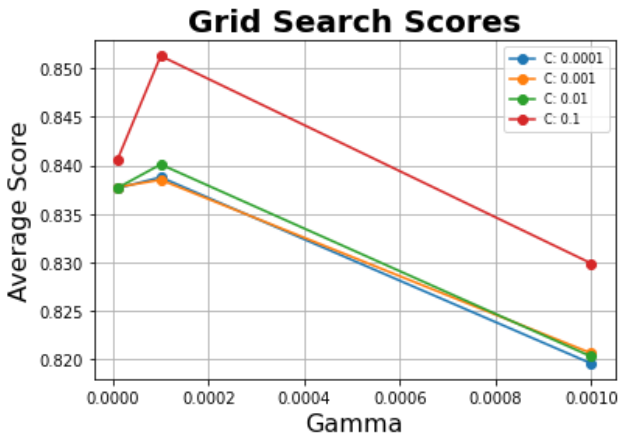

Figure 1: Illustration of the Hyperparameter results against the Average Score. (Case: StandardScaler effect)

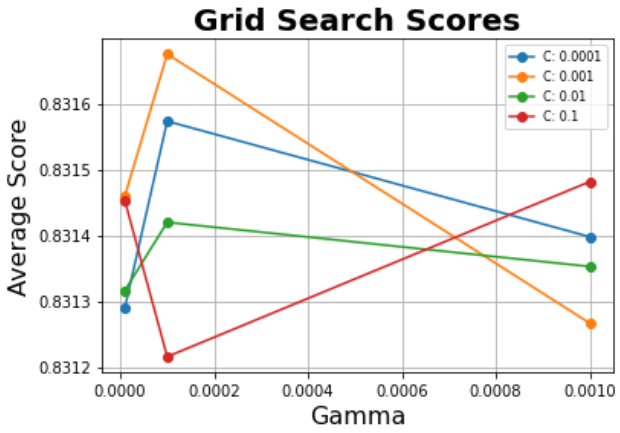

Figure 2: Illustration of the Hyperparameter results against the Average Score. (Case: Normalizer effect)

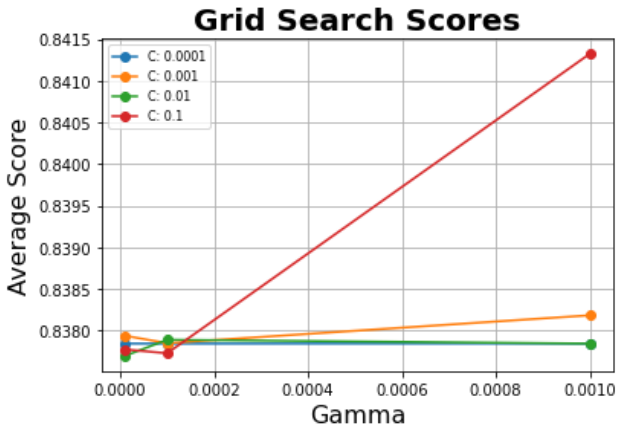

Figure 3: Illustration of the Hyperparameter results against the Average Score. (Case: MaxAbsScaler effect)

According to the above graphs, it is noticed that the best average ROC AUC score was equal to 85,1283% and obtained in the StandardScaler case, with C = 0.1 and gamma = 0.0001. This case achieved a score of 0.78311% in the 25% of the test data.

Also, it is evident that the average score, for all the cases, took its maximum value when C had the largest value: 0.1, which is logical since the error rate is smaller in that case. In addition, lower values of gamma seem to give higher results for the StandardScaler and Normalizer case, whereas, this trend changes on the MaxAbsScaler scenario.

4. Discussion

My overall goal for this project was to build a classifier and find ways to apply my knowledge to improve its effectiveness. Looking through what I have implemented, one big drawback is that I could not a find a way to inherit the specific imputation technique I used in the dataset, through the Pipeline. This might have led the dataset to data leakage hoping not in a proportion as other related pre-processing tasks would do. Also, instead of using the oversampling technique to balance the dataset, it might be a good idea to take advantage of the confidence rating of the labels that is given. The idea would be to swap the labels of the major class with low confidence rating, to the minority class. Another idea of how I could use the confidence rating in the future, would be to create a model where I could adjust the loss function to reflect the confidence scores, I have on the training data. Using a cross-entropy loss would help on that since it generates readily to comparing any two distributions. For instance, if an instance x that have a confidence 0.66 that the correct label for instance x is 0, then that would correspond to a probability distribution (0.66,0.44). Then, I could compute the cross-entropy of the classifier's prediction with respect to the distribution (0.66,0.44), which is the contribution to the loss from training instance x. Afterward, summing this over all instances in the training set, would get me an adjusted loss function.). Finally, the classifier would be trained by minimizing this adjusted loss function.

In conclusion, (SVM) classifier has been shown remarkable results in binary classification tasks, due to its strong ability of using the kernel trick to manipulate each form that is given as well as it scales relatively well to high dimensional data. However, choosing a “good” kernel function is not easy and considering its long training time for large datasets (e.g., like we have), it becomes a real problem. Also, C and gamma is hard to fine tuned as it is very hard to visualise their impact each time. Definitely, in a future related work, I would try to investigate the effectiveness of more models such as a Multi-Layer Perceptron (MLP), which also handles high dimensionality data, as well as more feature selection methods such as wrapper methods, which have been proved very promising (e.g., Recursive Feature Elimination (RFE)).

References

[1] Tang, Y. (2013). Deep Learning using Linear Support Vector Machines. Retrieved from http://arxiv.org/abs/1306.0239

[2] Agarap, A. F. M. (2017, December 10). An Architecture Combining Convolutional Neural Network (CNN) and Support Vector Machine (SVM) for Image Classification. ArXiv. arXiv.

[3] T.G. Stewart, D. Zeng and M.C.Wu., "Constructing support vector machines with missing data", WIREs Computational Statistic, vol.10, no. 4, 2018.

[4] Chawla, N., Bowyer, K., Hall, L., & Kegelmeyer, P. SMOTE: Synthetic Minority Over-sampling Technique. In International Conference of Knowledge Based Computer Systems, pp. 46-57. National Center for Software Technology, Mumbai, India, Allied Press, vol.16, 2002.

[5] "sklearn.preprocessing.StandardScaler." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[6] "sklearn.preprocessing.MaxAbsScaler." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MaxAbsScaler.html Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[7] "sklearn.preprocessing.Normalizer." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[8] "sklearn.preprocessing.SelectKBest" [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html

[9] "sklearn.preprocessing. f_classif." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html#sklearn.feature_selection.f_classifScikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[10]"sklearn.model_selection.cross_val_score."[Online]. Available: https://scikit-learn.org/stable/modules/cross_validation.html Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[11] Max Kuhn, Kjell Johnson "Applied Predictive Modeling" 1st ed. 2013, Corr. 2nd printing 2018 Edition

[12]"sklearn.pipeline.Pipeline". [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[13]"sklearn.model_selection.GridSearchCV". [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[14] "sklearn.metrics.roc_auc_score". [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.