Introduction to Machine Learning: Project assignment

Fares Meghdouri

November 26, 2019

Instructor: Prof. Dr. Dipl. Eng. Aggelos Pikrakis

1 Problem Statement

The problem that we solve in this work is a classical video labeling task using machine learning classification algorithms. We try to build models that can predict the label of a video based on previously seen data (training data). The data that we will use for this task is extracted from the Youtube-8m dataset by Abu-El-Haija et al. (2016) a massive dataset (around 8 million videos) with three variants of data representations.
We opt for a set of six classifiers: Naive Bayes, k-Nearest Neighbor, Least-Squares-based, Artificial Neural Networks and Support Vector Machines in order to predict the labels in a supervised learning manner (the training data is already labeled). Since the raw video representation (frame-by-frame) is hard to deal with due to the high dimensionality and structure of the data, we use the *video - level features dataset*. Once the classification framework is designed, a comparative study is carried out by evaluating each classifier using supervised learning metrics which allows us finally to conclude which model solves the problem better.
itFor the sake of simplicity, we skip all details regarding how algorithms work however, we encourage the reader to check the respective literature.

2 Data set Preprocessing

An important step in any Machine Learning (ML) application is data preprocessing. In fact, the way how we represent data is a key decision that reflects weather further operations will be successful or not. That is the reason why we invested time and resources making the data as clean as possible. We use the *Video-level features dataset* which is a version of the dataset that consists of two sets of features in addition to a label. The first set represents audio information consisting of 128 real values features extracted from the last layer of a CNN based architecture called VGG developed by Simonyan and Zisserman (2014), the network was trained on the dataset itself. The second set represents video information consisting of 1024 real valued features extracted from the bottleneck layer of an Inception-V3 image annotation model by Szegedy et al. (2016), trained on ImageNet. We get a total of 1152 features which is at a first glance a massive number with which we assume that most ML algorithms will either take forever to train or overfit.
We start by selecting only three classes from the dataset instead of the whole 3862 classes. By trial and error (with the goal of maximizing performance) and choosing three classes that intuitively seem different than each other (clusters far from each others in the input space) we ended up with the following classes: *Racing*, *Smartphones* and *Drum*. Table 1 show the IDs and the count of data simple in our sub-dataset.
Moreover, we show in Figure 1 and Figure 2 the accumulative distribution and the number of instances respectively. We note that the sub-dataset is almost balanced and, the train-validation is the same for all classes (80% training and 20% validation).
In order to extract the previous features we parse all tensorflow protocol buffer files (which is a special representation of data in order to store it physically) and for each, we first extract the label and compare it into our provided list of labels. If the former matches our choice, we parse the set of features and append them into an output file on the hard drive otherwise, just skip the record. The final representation of the data is a

ID	label	# instances in raw training set	Train set	Validation set
19	Racing	84258	84285	24182
23	Smartphone	64884	64893	18726
33	Drum	54195	54199	15705

Table 1: Dataset statistics.

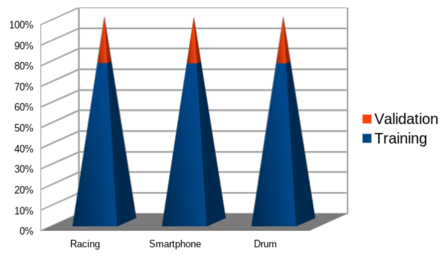


Figure 1: Relative Train-Validation distribution of each class in our dataset.

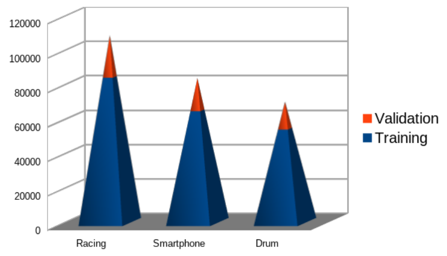


Figure 2: Train-Validation distribution of each class by numbersin our dataset.

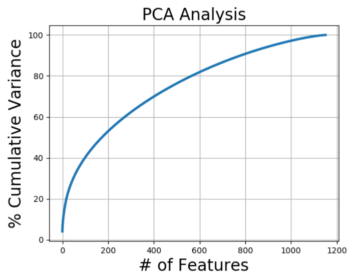
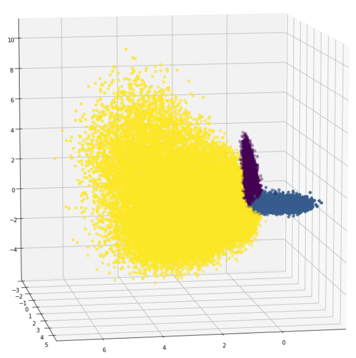
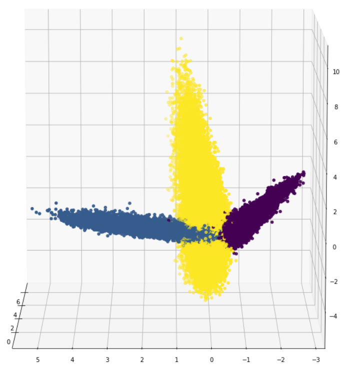


Figure 3: PCA analysis.

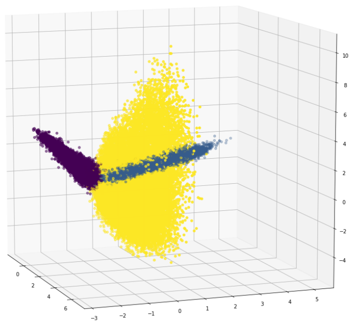
Comma Separated Values (CSV) containing 1152 features (columns) and a label. Files are eventually available publicly[1].



(a) Angle 1 of the input space.



(b) Angle 2 of the input space.



(c) Angle 3 of the input space.

Figure 4: The new input space after PCA. the three classes are easily separable.

We proceed next to the dimentionnality reduction, we chose two basic techniques:

- First, we convert the original space into a rotated space by maximizing the variance in the first few dimensions, this can be achieved by using Principal Component Analysis (PCA). Figure 3 show the accumulative variance distribution of all components. After experimenting, we noticed that the first 3 dimensions contains 15% of the information but represent the data perfectly. Figures 4(a), 4(b) and 4(c) show the reduced space and the colors represents the three classes. It is worth to mention the the data is scaled before performing PCA.
- The second option is Random Forests (RF) feature importance metric which is a measure for how pure a feature is (in other words, features with higher split in the tree get higher importance). Applying this technique and removing features with vanishing importance let us reduce the dimension from the raw dataset, however this technique is unexplainable, we tried it at first bu we opted only for PCA at the end.

One of the decisions we had to make here is the order of reduction/selection. We applied only PCA since the resulting three dimensions meet our needs and there is no need for further reduction.
Additionally, it is worth to mention that ANNs are able to learn the feature importance without the previous steps however, if we aim for a fair comparison, we feed the same data to all algorithms. Moreover, the higher the dimension is, the more risk we have to over-fit. Actually, in our case we tried with the full dataset at first but it turns out that all our models strongly over-fit hence, we decided to loos information for the sake of reducing over-fitting.
Moreover, the resulting three dimension are not generic meaning that other types/classes of videos might not be well represented in our space but for this task, the later should be sufficient.

3 Experiments

The framework used for the classification task is rather simple. Figure 5 show the different stages used in order to evaluate each classifier. We first start by feeding the scaled training set and training labels into the algorithm

[1] `https://owncloud.tuwien.ac.at/index.php/s/OB60wk49xzhj398`

under test. A 5-fold cross-validation scheme is applied in order to generalize the performance of each model by monitoring the accuracy of each fold. In addition, a parameter tuning step is optionally added in which the algorithm's parameters are tuned using evolutionary search for better performance (the evolutionary search uses genetic algorithms to get the best performing child). After performing the cross-validation, a fully working model is obtained utilizing the same previous architecture however this time by training it with the whole training data. The former model is then used to predict classes of **unseen** data sample from the validation set as well as data samples from the training set which results in two sets of predictions. Finally, we compute confusion matrices for both training and validation sets using the ground-truth provided with the validation set. We use three supervised learning metrics for the evaluation: Accuracy, weighted F1-score and weighted Jaccard index.
It is worth to mention that the labels were additionally preprocessed for a few algorithms. For instance, ANNs and multi-class Least Squares need a three dimensional target vector so the raw labels were converted into dummy variables, meaning that a single column of labels is converted into a multi-column binary output.
For reproducibility purposes, all scripts used for preprocessing, training and testing are available in the personal github repository[2].

3.1 parametrisation

We discuss here briefly the configuration of each algorithm.

Artificial Neural Networks The architecture used consist of an input layer with the same dimension as the input data, 9 neuron and 16 neurons hidden layers with *relu* activation functions and batch normalization and finally a three neurons output layer with a sigmoid activation (we used sigmoid here to obtain probabilities since we wanted to investigated the confidence). The choice of the number of layers and number of neurons is arbitrary. Moreover, we use 30% dropout after the first hidden layer to reduce the dependence of the network on some neurons since we expected a large over-fitting due to the lake of the training data. Moreover, we use 20 training epoch with a batch size of 32 and early stopping if the validation loss does not change by 10^{-6} for three epochs. For the back-propagation, we use an adam optimizer with the categorical cross-entropy loss function to train the network and we monitor the accuracy score after each epoch.

Naive Bayes For this classifier we use a Gaussian Naive Bayes without any parametrisation and without any prior probabilities. We furthermore, assume that the features are uncorrelated (since they were already extracted from a deep network and PCA were applied afterwards) which is an important assumption for a NB classifier to work.

Random Forests We tune two of the RF parameters: maximum depth of the tree and the minimum samples needed to create a leaf, we find respectively 12 and 6 after 6 generation and 300 combinations of a genetic search. Moreover, we use a set of 10 trees in the forest. In order to measure the quality of splits, we use the gini impurity.

Support Vector Machine The only parameter that we use in this classifier in the sigmoid kernel under the assumption that a linear kernel cannot separate the space perfectly. The penalty parameter is kept as 1 and we did not tune the parameters since SVM classifiers are known to be computationally heavy.

k-Nearest Neighbor We tune the algorithm and found that 10 nearest neighbors is the best parameter for finding the class of a new data point. The distance used is the euclidean distance, this again should be avoided in high dimensional data since the euclidean distance is proved to show unexpected behavior however, since we are dealing with three dimensions, it is fine to use it.

Multi-class Least Squares There were no parameters to tune in this case. Since the multi-class least squares classification is not implemented in well know libraries, we use a python implementation[3] and improve it. In general, the idea is to find the weights that is in this case not a vector but a matrix that contains three dimensions, one for each class. Later on, the class of a new data sample $\underline{x}$ is estimated by computing $\widehat{\underline{y}} = argmax \widehat{\mathbf{W}}^T \underline{x}$ and taking the argument of the maximum value.

[2]`https://github.com/fm94/itmlwa`
[3]`https://github.com/HarryGogonis/Python-Linear-Least-Squares-Classifier`

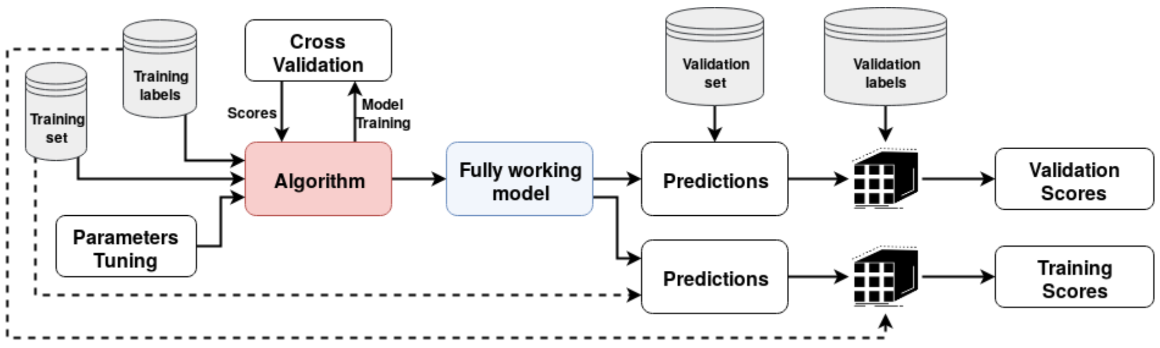


Figure 5: Analysis framework.

Data	ANN	NB	RF	SVM	kNN	LS
Training Set	0.967	0.929	0.968	0.905	0.970	0.846
	(+/- 0.006)	(+/- 0.001)	(+/- 0.001)	(+/- 0.002)	(+/- 0.001)	(+/- 0.004)

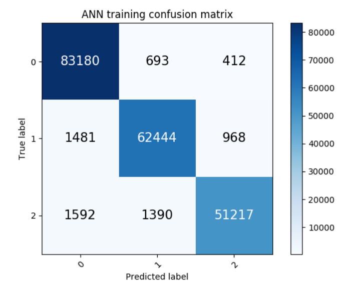
Table 2: Cross-Validation accuracy scores.

4 Results and Discussions

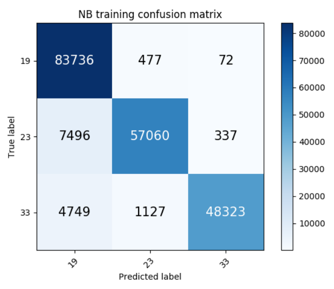
We start by observing the cross-validation scores of the training set. Table 2 show the average accuracy score with the standard deviation of each algorithm. As a baseline, the majority random guessing accuracy score is 70% which is the best score that we can get randomly.
Interestingly, KNN perform the best during the training cross-validation with even a low standard deviation. ANN, RF performed similar and slightly worst than KNN, NB show lower performance followed by SVM and finally LS which was expected since the problem is not totally linearly separable. The NB classifier was expected to perform worst however the fact the PCA was applied before classification, the former reinforced the assumption of independent features which improves the performance of the NB classifier.
We move on the the confusion matrices, time time we show a set of twelve confusion matrices representing six algorithms and both training and validation sets in Figure 4.
A few things that we can observer from the confusion matrices are:

- The first phenomena to notice is the diagonality of the matrices. The former reflect a collection of good classifiers since a diagonal confusion matrix means less False Positives and False Negatives. However, in any case, many samples were miss-classified in the case all algorithms.
- The second phenomena is that the training set always performs better that the validation set. This is quite intuitive since any algorithm is able to better classifier data samples that it has seen before. Despite the fact that predicting classes of the training data is useless in practice, the former gives us an idea on the generalization of our model.
- The third phenomena that we notice is that RF performed extremely well on the training set but the poorly for the validation set. This is a well known property of RF since during training the trees tend to over-fit if the the maximum depth is large (12 in our case). As mentioned before, we opt for the cross-validation to reduced the over-fitting.
- The last phenomena that we notice is triangular confusion matrix in the case of LS classification. The explanation is rather strait forward: the hyper-planes created by the LS estimation moves toward the larger cluster (yellow) in Figure 2 thus, all misclassified instances lay only on one half of the confusion matrix. This exactly meets the theory discussed during the course.

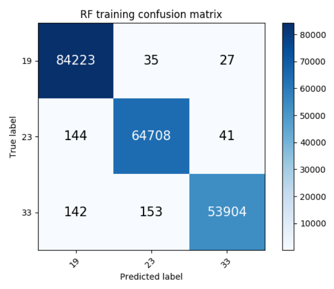
Finally, we show three metrics extracted from the previous confusion matrices for both the training set and validation set in Figure 7 and Figure 8 respectively. The previous conclusions hold here as well, RF performed extremely well on the training set but poorly on the validation set where KNN won. LS show the worst performance.
If we would make a list of the best classifiers based on the validation set then: **(1) KNN, (2) RF, (3) ANN, (4) NB, (5) SVM, (6) LS.**



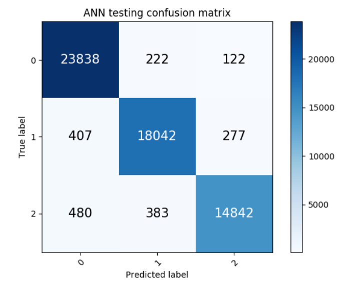
(a) Training set performance with an Artificial Neural Network model.



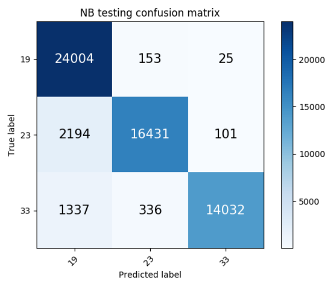
(b) Training set performance with a Naive Bayes model.



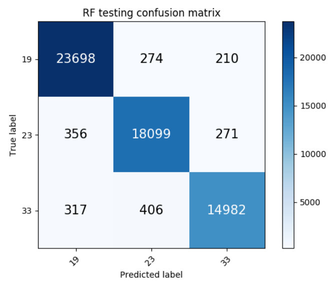
(c) Training set performance with a Random Forests model.



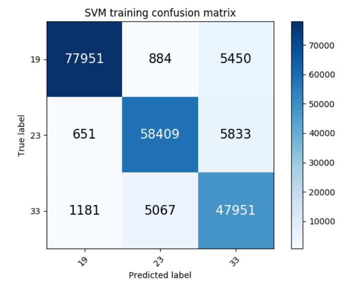
(d) Validation set performance with an Artificial Neural Network model.



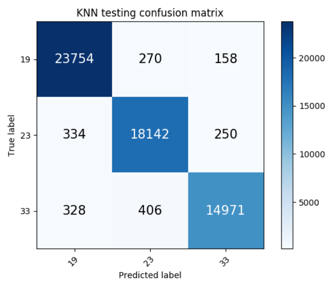
(e) Validation set performance with a Naive Bayes model.



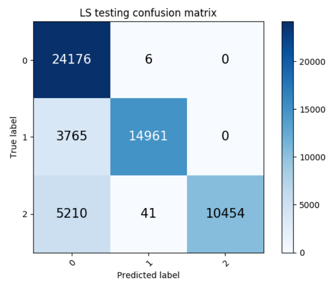
(f) Validation set performance with a Random Forests Network model.



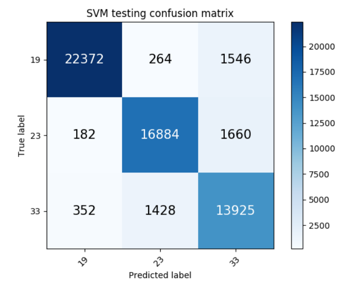
(g) Training set performance with a Support Vector Machine model.



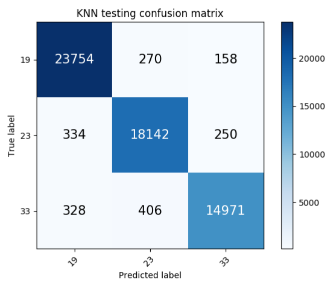
(h) Training set performance with a K-Nearest Neighbor model.



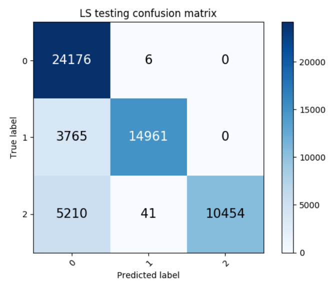
(i) Training set performance with a Least Squares model.



(j) Validation set performance with a Support Vector Machine model.



(k) Validation set performance with a K-Nearest Neighbor model.



(l) Validation set performance with a Least Squares model.

Figure 6: Confusion matrices of different models.

5 Conclusion

In this work, we created a benchmark to compare the performance of six classification algorithms for predicting videos labels in a supervised manner. We filtered out three classes from the well known Youtube-8M dataset. Moreover, in order to reduce the dimensionality, we opt for PCA feature reduction technique and reduced the dimension to only **three** dimensions. In order to evaluate our algorithms, a cross-validation in addition to a comparative study is carried out. Three metrics (Accuracy, F1 score and Jaccard score) were extracted from the confusion matrices and compared.
Results suggest that most models performed extremely good except the Naive Bayesian approach. However, we believe that the classes that were chosen (Racing, Smartphones and Drum) are easily separable and far from

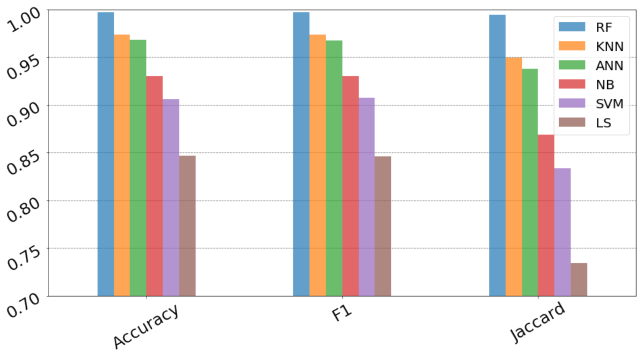


Figure 7: Training set scores.

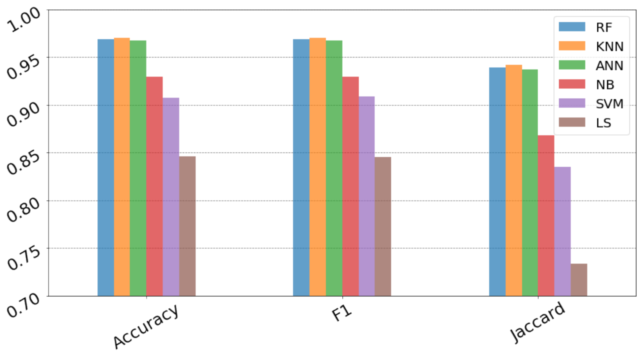


Figure 8: Validation set scores.

each other in the input space, this is one of the reasons that allows us to reduce the dimension of the input space and later on led us into high prediction scores. In fact, we tried many classes combinations before with which the performance were worst than a random guessing classification.
The fact that features were extracted from well known powerful architectures let us assume that the data is already well represented and for sure, dealing with raw videos would be way much harder.
We note that in earlier experiments with much more dimensions, all classifiers over-fitted. This is obvious since the larger the dimension is, the more training data we need so we tried to make the problem harder by removing considerable amount of information by applying PCA.
Finally, we prove that no matter which approach one takes to solve such real problems, with the same complexity the results should almost always be similar and no magic is included in the formula.

References

Abu-El-Haija, S., Kothari, N., Lee, J., Natsev, P., Toderici, G., Varadarajan, B., and Vijayanarasimhan, S. (2016). Youtube-8m: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675*.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.