CS 181 Practical

Jothi Ramaswamy, Justin O'Dwyer
jramaswamy@college.harvard.edu, justinodwyer@college.harvard.edu

May 7, 2022

---

1 Part A: Feature Engineering, Baseline Models

1.1 Approach

What did you do? When relevant, provide mathematical descriptions or pseudocode. Credit will be given for:

- PCA: Describe what the top 500 principal components represent, and how you computed them.[1]

Our first step in this process is to perform principal component analysis on both datasets. The idea behind PCA is that there are certain dimensions of the data that are much more important than others when it comes to explaining variances in the data. In order to identify the order of importance of these dimensions, we can calculate the empirical covariance matrix as

$$\mathbf{S} = \frac{\mathbf{X}^T\mathbf{X}}{n}$$

where $\mathbf{X}$ is the design matrix.[1] We can then study the empirical covariance matrix $\mathbf{S}$ and obtain its eigenvectors, with the eigenvectors ranked from highest to lowest eigenvalue corresponding to the principal components ranked from greatest to least importance with respect to explained variance. After performing PCA as described above on both the raw amplitude features and the Mel spectrogram features, we obtain the 500 most significant principal components. The top 500 principal components represent the 500-dimensional subspace that we can project our data down to while losing the least amount of information from our data. They also represent the top 500 vectors that explain the directions of maximal variance in our dataset. We performed PCA because we wanted to reduce the dimensionality of the features in our datasets significantly — both the amp and mel datasets have over 10000 features, which can make modeling our data very tricky. This is because many of these features are noisy and can make it harder to find meaningful trends in some important features of the data. To do this efficiently, we imported a package from scikit learn.

[1] The formula for the empirical covariance matrix can be found in page 97 of the CS 181 textbook.

- Logistic regression: Describe how the model you trained predicts output probabilities for each class.

After reducing the dimensionality of the data into something that is easier to work with through PCA, we then proceed to train a logistic regression classifier that will classify sounds into one of the ten given categories.

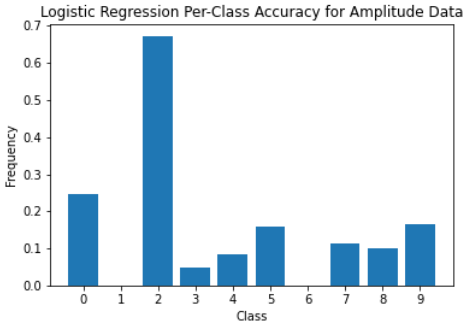
In order to do this, we create a function $h_k(\mathbf{x}_n) = \theta_k^T \mathbf{x}_n$ where $k \in \{0, \ldots, 9\}$. We then create a vector $\mathbf{z} = [h_0(\mathbf{x}_n), \ldots, h_9(\mathbf{x}_n)]$ and pass $\mathbf{z}$ into a softmax function, which will then output a nine-dimensional vector of probabilities, each denoting the probability that the given example falls into a particular class. We then define the loss function as the negative log likelihood of the data, and we then take the gradient of the loss function which we can use to perform gradient descent. For the purposes of implementation, we use the LR function from sk learn.

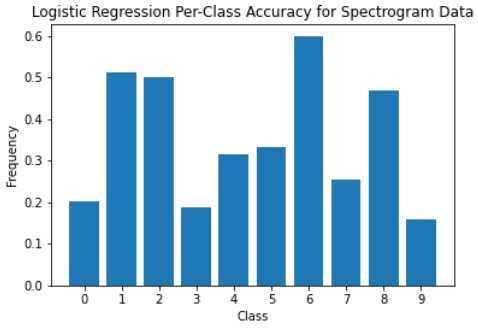
1.2 Results

This section should report on the following questions:

- What is the **overall** and **per-class** classification accuracy of the models that you implemented?

The overall accuracy of the logistic regression model on the amplitude data is a little under 21%. In contrast, the overall accuracy of the logistic regression model on the Mel spectrogram data is about 31%.





Logistic Regression Overall and Per-Class Accuracy for Raw Amplitude and Mel Spectrogram data

Type	Overall	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
Amp	0.203	0.233	0.000	0.649	0.044	0.091	0.178	0.000	0.102	0.102	0.180
Mel	0.310	0.203	0.513	0.502	0.188	0.314	0.333	0.600	0.254	0.470	0.160

1.3 Discussion

This section should report on the following questions:

- Why do you hypothesize one feature representation performed better than the other?

From the practical instructions pdf, we know that the Mel spectrograms partition the amplitude arrays into subsequences and compute the presence of a range of frequencies across these subsequences. Given that the explained variance from PCA is much higher for the spectrograms than for the amplitude measurements, it seems very natural that the spectrogram feature representation would perform better in tests, and we hypothesize that this is due to the transformation of the data into the frequency domain which may be a better representation of the data for this problem.

- Why might have asked you to perform PCA first, and what is the impact of that choice?

Because the data is extremely high dimensional and the dataset is large, using PCA allows us to avoid having to partake in a very computationally expensive procedure while also shrinking the sample space dramatically. Because the data is extremely high dimensional, the data is likely to be much more sparse leading models to have a more difficult time generalizing due to the curse of dimensionality.

2 Part B: More Modeling

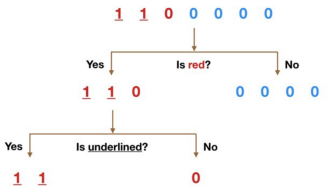
2.1 First Step

2.1.1 Approach

What did you do? Credit will be given for:

- Provide mathematical descriptions or pseudocode to help us understand how the models you tried make predictions and are trained.

Our initial approach for this problem was to use random forest classifier. A random forest is an ensemble learning method that is used for classification by constructing a number of decision trees, each of which "votes" for a class prediction for a given input, with the class receiving the largest number of votes being the predicted class. Each individual tree might look something like the image below.



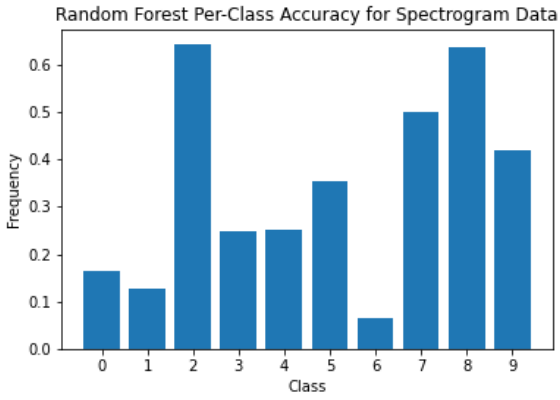
The random forest method then creates a bootstrap dataset from the original dataset and then proceeds to randomly choose certain features in order to construct a decision tree. This process is repeated a large number of times in order to have a wide variety of trees, and the randomness that comes with the generation of these trees and the features chosen to construct them cause the voting process to be often quite successful in predicting classes. In terms of the hyperparameters we chose, we set the maximum possible tree depth to be 10 and the number of trees in the random forest to be 500. We chose 10 for the max tree depth because I didn't want the model to overfit the training data and have an awful test score. We chose 500 trees because we thought it could be enough trees, and significantly more than the default of 100, to have robustness against any outlier trees. Having this robustness to outliers is really important in our case because we have a huge dataset and the subsets of the data used for some trees can be very different from the subsets of data used for other trees.

2.1.2 Results

This section should report on the following questions:

- What is the overall and per-class classification accuracy of the models that you implemented?

The overall accuracy of the random forest model on the Mel spectrogram data is about 39%. In contrast, the overall accuracy of the logistic regression model on the Mel spectrogram data is about 31%. Below are the per-class accuracies for the random forest model:



Random Forest Overall and Per-Class Accuracy for Mel Spectrogram data

Overall	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
0.391	0.15	0.154	0.652	0.231	0.269	0.364	0.067	0.496	0.623	0.427

2.1.3 Discussion

Compare your results to the logistic regression models in Part A and discuss what your results imply about the task.

2.2 Hyperparameter Tuning and Validation

2.2.1 Approach

What did you do? Credit will be given for:

- Making tuning and configuration decisions using thoughtful experimentation. How did you perform your hyperparameter search, and what hyperparameters did you search over?

When we were deciding how to tune our random forest models, two hyperparameters really stood out as being really important: the number of trees we averaged together (n_estimators) and the maximum possible depth of the tree (max_depth). The number of trees we averaged together is really important since there are tradeoffs if we make this number too high or low. Adding more trees to the model essentially provides robustness against any one outlier tree that could skew the model and result in poor performance. However, if we provide too many trees such that our marginal value of adding another tree is very low because we are already robust against outliers, our model becomes extremely slow, so we want n_estimators to be right in the middle.

We also want to test out values of max_depth as well. If max_depth is too high, we become vulnerable to overfitting the data and having trees that are too specific to the training set. However, if max_depth is too low, we have the opposite problem and will end up underfitting our data with a model that is too simple.

We performed our hyperparameter search using tools from sklearn's GridSearchCV which takes all the possible combinations of the hyperparameters we tune and performs 5-fold cross-validation on each of these models. Our possible hyperparameter selections include n_estimators = [100, 300, 500, 700, 900] and max_depth = [10, 20, 30, 40, 50].

2.2.2 Results

Present your results of your hyperparameter search in a way that best reflects how to communicate your conclusions.

The model that performed the best was the model with a max_depth of 20 and an n_estimators equal to 500. This model had an accuracy of almost 43%. The rest of the models' accuracies can be seen below. While these accuracies are all extremely close to each other, we can still see some trends in the resulting rankings.

Scores of random forest models trained on Spectrogram data with different hyperparameter combinations

param_max_depth	param_n_estimators	mean_test_score	rank_test_score
20	900	0.429492	1
50	900	0.429312	2
40	900	0.428772	3
40	500	0.428592	4
50	700	0.428233	5
30	900	0.428053	6
20	500	0.426791	7
20	700	0.425533	8
40	700	0.425353	9
30	700	0.424092	10
50	500	0.422832	11
30	500	0.420127	12
30	300	0.418870	13
20	300	0.418689	14
40	300	0.418329	15
50	300	0.418327	16
20	100	0.409322	17
50	100	0.406264	18
40	100	0.404282	19
10	500	0.403382	20
10	900	0.402664	21
10	700	0.401941	22
30	100	0.400681	23
10	300	0.397621	24
10	100	0.388615	25

2.2.3 Discussion

Why do you expect the tuned models to perform better than the baseline models and the model used in First Step? Discuss your validation strategy and your conclusions.

Our validation strategy, as described above, was using 5-fold cross validation for each combination of hyperparameters. This seemed like a reasonable strategy because we have some robustness to outlier test scores, but we aren't training too many models per hyperparameter combination such that its taking too much time.

Looking at the table above, we can see a few underlying patterns. In terms of the max_depth, having a depth of at most 20, 30, 40, and 50 seemed to perform similarly and there wasn't much of a distinction between them, so it doesn't seem like a depth of more than 20 was really needed. Having a max_depth of 10 however seemed to mostly perform the worst in terms of the test score, so it seems like the models with a max_depth of 10 were underfitted. In terms of n_estimators, we see a pretty clear general trend of a better performance with more trees in the random forest, which

is as expected because we have more robustness against outlier trees as we increase the number of trees we average against.

Overall, we would still expect the tuned models to perform better than the baseline models and the models used in the first step because we try out a variety of possible combinations of hyperparameters and end up with a combination that addresses over/underfitting the most. It makes sense that the tuned models at the top of the list performed better than the model we tried in part B1 because the hyperparameter combination we started with resulted in the model underfitting the data slightly because it restricted the tree depths to 10 and only used 500 trees. However, the best trees when we tuned the models had more trees and robustness to outliers and didn't restrict the tree depth as much. In part B1, we were only really concerned with not overfitting the data, that we ended up with the exact opposite problem of underfitting the data.