Classification of bird species based on recorded bird-bird-soundsounds

Pritam Sukumar

October 13, 2016

Contents

1 Definition

A friend and I are building an app that will use machine learning to recognise bird sounds from iPhone recordings, and I wanted to use this project as an opportunity to explore work that has already been done in this area. While researching potential topics related to bird sound recognition, I found a Kaggle competition hat deals with the same exact problem along with a potential approach that I could try out. I decided to go along with the steps and implement the machine learning portions of the competition myself to learn more about this field.

1.1 Project Overview

The recognition of bird sounds holds importance for a variety of reasons. Identifying the bird species in a given region allows scientists to prepare surveys of current populations, along with aiding studies of specific species, investigating the effect of environmental factors such as climate change, and also providing data for making predictions about the future in that area. Also, the hobby of bird watching has grown immensely, and for bird watchers, spotting a new species (or even a known one) is a very pleasing experience.

Identifying birds present in an area for scientific research is an expensive task, both in terms of time and money, due to the necessity of in-the-field researchers. For hobbyists, it has always been a challenge – and, to be honest, part of the charm – to recognise new species in the wild. In recent years, there has been a strong interest in the possibility of using machine learning for this task.

One approach that was followed (and is explored in this project), is using data from microphones placed in the forest. This has the advantage of not needing human presence to record the sounds. The microphones are retrieved every few weeks and the data is used to identify the bird species present in the recording. In this project, I am going to try to (or rather verify) that it is indeed practical, and that the accuracy that we can obtain using relatively new data-processing and machine learning techniques is very promising for the future of bird-sound recognition and my app.

1.2 Problem Statement

The scope of this project is defined by the scope of the The task is to use machine learning to automatically detect the species of bird in a recording made in the wild. The recordings are usually of short duration. However, they may contain no birds (as is the case for most of the day), or multiple birds in the same recording (as is often the case for recordings taken at dawn or dusk).

The strategy applied for this task is to understand how to extract information from the sound recordings. This usually entails the following steps:

1. Detecting whether bird sounds exist in the wave file
2. Separating the bird sounds from the background noise (streams, rain, etc.)
3. Detecting whether there are one more more birds
4. Separating the individual bird sounds
5. Processing the bird sounds in a way that is friendly to machine learning classification

I intend to retrace the steps of a paper that accompanies the Kaggle competition. The first step is generating a spectrogram, i.e. converting the audio file to the frequency domain because birds vary much more by pitch than by amplitude (which is what a usual wave file will provide). Once that is done, we reduce the noise in the data, and then separate out the bird sounds from the rest of the audio (the segmentation problem), and also from each other (the cocktail party problem).

Then we can run a classifier over the segmented frequency graphs to detect the birds in a given recording. Note that the problem complexity increases because this a Multi-Instance Multi-Label (MIML) problem, i.e. Every audio file can have multiple bird sounds (instances) in it, and each instance of a bird sound can correspond to one of many different birds (labels).

In this project, I try four different classifiers to compare their performance: Decision Trees, K-nearest neighbours, Random Forest Regressors, and Logistic Regression. The working of these algorithms is briefly explained in Section **??**.

The final result should be something along the lines of: "In recording m, there are three birds present, which are x, y, and z."

1.3 Metrics

Since I am using a dataset from an expired Kaggle competition, the metrics are defined by the rules of the competition. In this case, the metric is the overall accuracy of the classification as measured by the area under the ROC curve. As defined by the competition, the ROC curve is generated by taking as input the labels generated by the algorithm, along with the predictions.

The ROC (Receiver Operating Curve) is a curve that plots the true positive rate versus the false positive rate for a binary classification algorithm. The AUC (Area Under the Curve) represents the accuracy of the algorithm.

In the case of the bird-classification problem, since the problem is multiclass, we need to first convert it to a binary classification problem. We do this by simply treating the presence of each bird in each recording as a separate instance for the algorithm. This way, every sound recording corresponds to nineteen classifications.

Reframing the problem as a binary problem (Binary Relevance) has the disadvantage of biasing the accuracy metric towards true negatives, since there is a higher probability that a bird will not be present in most of the recordings. Since the ROC curve is plotting the true positive rate versus the false positive rate, each point on the ROC plot effectively represents *two* instances, one each from the correct and incorrect classification groups. Thus, if every example were marked as absent, the AUC would be 0.5.

2 Analysis

In this section, I present an analysis of the specific problem at hand, including a description of the dataset, the approach followed to solve the problem, an explanation of how the audio recognition problem is transformed into an image recognition problem, and finally, the specific machine learning techniques I tried to use for this problem, along with reasoning for them.

2.1 Data exploration

The training set consists of 323 wave files, representing sounds from 19 species of birds (see Table **??**). Each recording contains up to 7 species of birds making sounds in them, maybe simultaneously.

Table 1: Birds present in the dataset

Code	Name
0	Brown Creeper
1	Pacific Wren
2	Pacific-slope Flycatcher
3	Red-breasted Nuthatch
4	Dark-eyed Junco
5	Olive-sided Flycatcher
6	Hermit Thrush
7	Chestnut-backed Chickadee
8	Varied Thrush
9	Hermit Warbler
10	Swainson's Thrush
11	Hammond's Flycatcher
12	Western Tanager
13	Black-headed Grosbeak
14	Golden Crowned Kinglet
15	Warbling Vireo
16	MacGillivray's Warbler
17	Stellar's Jay
18	Common Nighthawk

The dataset is quite unbalanced with respect to each of the species, as shown in Figure **??**. In the region where the recordings were made, it looks like Swainson's Thrush was pretty common (more than 60 examples), while many other birds are represented by less than 10 recordings. This means we need an algorithm capable of handling this kind of unbalanced dataset for a multiclass problem.

2.2 Exploratory Visualization

As mentioned briefly in the preceding section, each recording can contain more than one recording, or even none. The presence of multiple birds in the same recording presents great challenges in terms of segmentation and source separation (the so-called cocktail party problem). The presence of more than one bird-sound with the possibility for overlap indicates that we will have to do some processing to the recordings to be able to separate the bird-sounds from each other.

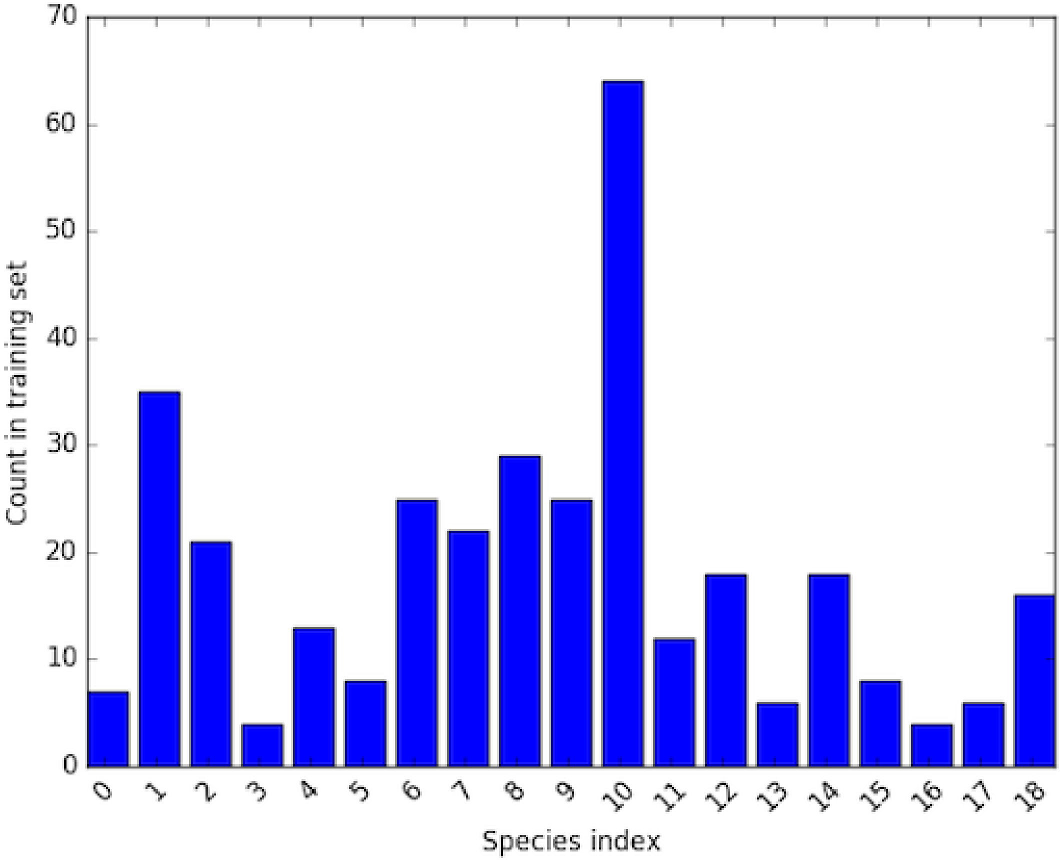


Figure 1: Number of examples in training set for each bird species

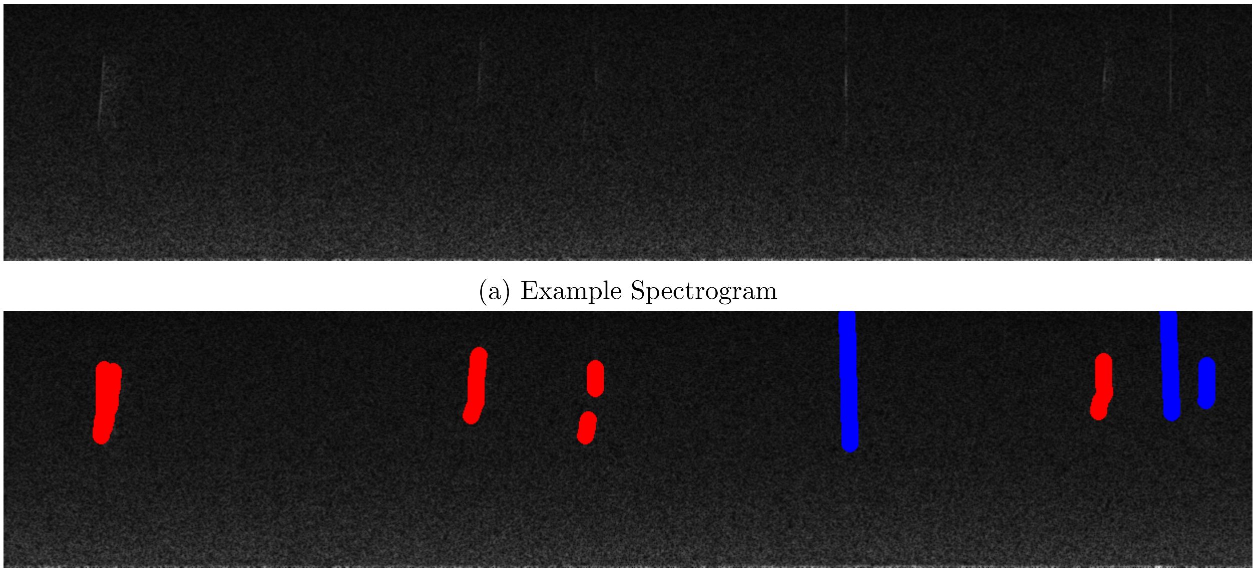
2.3 Algorithm and Techniques

To automatically classify bird-sounds, the sound recording needs to be converted to a form that is suitable as input to a machine learning algorithm. We first need to separate the bird sounds from the noise. There are two approaches commonly used to do this. Both of them rely on first extracting the spectrogram of the audio from the audio file.

In this section, I will first discuss how the authors of the paper obtain the spectrogram, following which I will discuss the machine learning algorithms I chose to study in addition to the ones used in the research paper.

2.3.1 Spectrograms and audio recognition

A spectrogram is a visual representation of the frequencies present in an audio file. In the spectrograms considered in this paper, the X-axis represents time and the Y-axis represents the frequency. A point *(x,y)*, represents the amplitude of frequency y at time x. The amplitude is usually represented by a shade of white ranging from 0 (black) all the way to 255 (white). Black indicates that the frequency is not present at that time, and white indicates that the frequency is present at high amplitude (i.e. loud) at that point of time, with shades in between filling in the range.



(a) Example Spectrogram

(b) Spectrogram after noise reduction and segmentation

Figure 2: Example spectrograms provided along with the competition dataset

Most recognition problems (and work in many other areas) depend on a spectrogram since the frequencies present in an audio are, in fact, the things that define it. Also, using the frequencies allows us to automatically account for the effect of loudness since the frequency does not depend on loudness.

For sound recognition, there are two approaches that are popular:

1. Get a vector descriptive of the frequency range in the spectrogram and use that for recognition. One way to do this is convert the spectrogram to a *Mel-Spectrum*, i.e. apply a bunch of filters spaced logarithmically over the range of human frequencies to get a set of vectors for each of the syllables and/or sounds present in the audio. The vector can then be used as an input to a Machine Learning algorithm.

2. Use the spectrogram as an input to an image recognition problem, thus enabling the use of techniques such as segmentation and histogram-of-gradients. We can separate out sounds using segmentations and then each sound can be fed into a machine learning algorithm for classification.

2.3.2 Generating features from the segments

The approach followed by the Kaggle competition organisers to generate recording-wise features follows closely the approach outlined in the 2012 paper by Briggs et. al.[**?**]. Of the two approaches outlined above, the authors followed the second approach, i.e. using the spectrogram as input to an image recognition problems. The steps in the process are:

1. *Noise Reduction*

2. *Segmentation*: Once the spectrogram is generated, a segmentation algorithm is run on the audio files to separate the individual bird sounds from the audio. The segmentation algorithm

is trained using 20 of the wave files that have been manually segmented by humans, and then a Random Forest is used on the rest of the data to separate the segments.

An example spectrogram of a recording and the result after noise reduction segmentation on the same recording are provided in Figures **??** and **??**. The red segments indicate non bird-sounds (for example, wind or trees or rivers) and the blue segments indicate a probable bird sound.

3. *Segment-wise features*: After the segments are separated, a histogram-of-gradients feature is used to generate a feature vector for each of the segments.

4. *Bag-of-words feature for each recording*: Note that the number of segments in an image may vary, so applying a machine learning algorithm to the segment features directly will be cumbersome. Instead of doing that, we generate bags-of-features from the histograms. Bags-of-features (commonly called bag-of-words) is a technique commonly used in text processing that provides information with how often a particular feature occurs in an instance.

   Basically, they are a way to accumulate information from different features into a single training instance. The output is a single fixed-length feature vector that can then be used for classification in a MIML problem. The final histogram-of-gradients features are provided along with the competition dataset and are 100x1 vector, and are what I used as input to my learning algorithm.

2.3.3 Algorithms for classification

It is not obvious immediately which machine learning algorithm would be most-suited to this task, so I decided to try a variety of algorithms:

1. Decision Trees (DTs): Decision trees reframe the classification problem as a tree, where at each node, a branch is taken based on a feature property. The properties of the tree are found using an iterative adaptive boosting algorithm, the most common of which is the AdaBoost. The advantage of decision trees are that they are partially immune to noise and the presence of outliers. However, trees are prone to overfitting, sometimes resulting in deep trees that perform well on the training set and poorly on the testing set.

2. Random Forest Regressor (RFR): This is the algorithm used in [**?**]. Simply put, random forests are collections of decision trees, trained on different parts of the training set [**?**]. Each tree votes for the final class depending on its output, and hence the final classification is the mode of all the classifications provided by each of the trees. The main advantage of RFRs over simply using a decision tree is that because we are considering multiple trees, each trained on a subset of the data, and a subset of the features, there is zero possibility of overfitting.

3. K-nearest neighbours (KNN): KNN is a popular algorithm used for classification problems. The final classification is obtained by finding the k nearest neighbours from the training examples, as measured by finding the “distance” between features, where distance can be arbitrarily defined, but is usually taken to be the euclidean distance. Once the k nearest neighbours are found, the final classification is again taken to be the mode of the classes of the training examples.

4. Logistic Regression (LR): Logistic regression was designed to only apply to single class classification problems. However, it can easily by extended to multi-class problems. If there n

classes, n sets of regression coefficients are found based on the training data. This is effectively the same as finding n separate LRs. Thus, when a test data point is encountered, the multiclass LR outputs a separate probability for each of the classes. The sum of these probabilities is 1, since the test data point is assumed to belong to one and only one of the classes. The predicted class is then, the class that has the maximum probability.

2.4 Benchmark

If we do not use any machine learning, and instead predict that there are no birds in any of the audio files, or predict the presence or absence perfectly at random, the area under the ROC curve will be **0.5**.

The benchmark for this project was the score provided by the organisers of the competition: **0.85576**. This score is generated by treating each recording as a collection of multiple single-label classification problems. A **Random Forest** is used on the bag-of-words features to generate this score. Once the presence of each species in each recording is predicted, this is used to generate an Receiver Operating Characteristic (ROC) curve. The ROC curve is a plot of the true positive rate versus the false positive rate for the algorithm. The benchmark score is generated by measuring the area under the ROC curve.

The goal of this project, since I am using the same features as the paper, is to meet this benchmark by using Random Forests, and to compare this with other machine learning algorithms to see if there is any difference.

3 Methodology

3.1 Data preprocessing

No preprocessing was needed as the complex transformations from the raw audio files to the filtered spectrograms was provided by the organisers of the competition. The feature vectors were extracted from the raw text files and then split into test and training sets. The code for this can be found in `data_extract.py`.

3.2 Implementation

I have outlined the algorithms I tried in Section **??**. Using the bag-of-words features provided with the dataset, I performed the classification by implementing these algorithms using the corresponding `sklearn` packages. Some of the challenges I faced were:

1. *Converting the multiclass features to a binary relevance problem*: The features as given, each correspond to multiple bird recordings. To treat the problem as a binary relevance problem, these need to be converted to a binary problem, i.e. for each recording, instead of saying: *Species (1, 5, 6) are present*, we need to say *Species 1 is present, Species 2 is absent, ...,* and so on. After spending many hours in this, I found that `sklearn` supports this out of the box with a package called **MultiLabelBinarizer**. The code for extraction is included in Listing **??**

2. *Performing the classifications*: I wanted to try multiple algorithms with multiple sets of parameters. However, the scoring was manual, since all the scoring was done through the Kaggle scoreboard. To get around this, I created a dictionary of *classifers* and then looped through them, fitting on the training data and classifying on the testing data. The results

were stored at runtime in separate `csv` files which could then be uploaded to Kaggle. The code for classification is shown in Listing **??**. Lines 11-37 set up the classifiers.

3. *Performing the grid search*: `sklearn` includes a method to perform grid search. However, the grid search does not automatically support multiple grid values. Hence, I simply manually performed the grid search, by defining nine separate RFR classifiers. The parameters were chosen to explore ranges around the parameters chosen by the authors of the paper this work is based on. [**?**]. Lines 11-33 of Listing **??** show the nine different classifiers used to perform the grid search.

3.2.1 Parameters

To start with, the default parameters were used for each of the algorithms For the sake of completeness, they are described below:

1. Decision Trees: The `sklearn` implementation of DTs includes many parameters to tune [**?**]. The most important ones are:
   (a) *Criterion*: The measure that is used to decide on which branch to take at each node. The `sklearn` DT package supports Gini impurity and entropy. The Gini entropy is used in this section
   (b) *Maximum features*: The maximum number of features to consider in the tree. This parameter is usually modified in order to minimize overfitting. For our purposes, we leave this value at the default value, which is to use all the features.
   (c) *Maximum depth*: The depth at which to stop growth of the tree. We do not set this value, which allows the tree to grow until all the final leaves are classes.

2. Random Forest Regressors: The `sklearn` documentation for RFRs [**?**] includes the following parameters (in addition to others) to tune:
   (a) *Number of estimators*: The number of decision trees constructed by the algorithm.
   (b) *Maximum features*: The number of features to consider when making a split at each node of each tree.
   (c) *Random state*: The seed for the random number generator used by the algorithm.
   (d) *DT parameters*: Parameters that are passed to the individual trees; these parameters are the same as those used by DTs.

3. K-nearest neighbours: The `sklearn` documentation for KNNs [**?**] includes the following parameters to tune:
   (a) *Number of neighbours*: The number of neighbours to consider for the final classification. The final classification will be the mean of these neighbours weighted by a function that is the next parameter. The default is set at 5, and that is the value we use for our analysis.
   (b) *Weight*: The weight to give each of the k neighbours. The examples can be weighted by distance from the centroid, uniformly or by a user-defined function. The default value is to use a uniform weight, i.e. to weight all k neighbours equally.

Algorithm	**Score**
Decision Trees	0.66217
Logistic Regression	0.74628
K-nearest neighbors	0.73433
Random Forest Regressor	0.78726

Table 2: Performance of algorithms with default parameters

4. Logistic Regression: The `sklearn` documentation for Logistic Regression [?] tells us that the developers have kindly allowed us to change the following parameters (actually more than just these, but I am only describing the relevant ones):

   (a) *Penalty*: The loss function used by the algorithm. By default the L_2 norm is used.

   (b) *Multiclass formulation*: The `sklearn` LR formulations supports two ways of approaching multiclass problems: as a multinomial problem, where a joint loss function is minimized, or as a binary problem, where separate LRs are fit for each class and the final classification is the label with maximum probability. We use the binary formulation, because that is what we are using with the other algorithms as well (I discuss this again briefly in the Future work section).

From Table **??**, it is clear that RFRs perform the best. RFRs are also the choice of the authors of the paper the Kaggle competition was based on [?]. This is probably not a coincidence. In the next section, I report on my work on refining the RFR with the hope that the scores improve.

3.3 Refinement

Among the algorithms, the Random Forest Regressor performed the best by far. To investigate further, I decided to investigate the performance of the algorithm with varying two parameters: the maximum number of features, and the number of estimators (trees). I performed a manual grid search (the `sklearn` grid search module does not support MIML problems) and improved the score of the algorithm. These results are tabulated in Table **??**.

The parameters that were obtained in the work of Briggs, et. al [?] are the ones in bold in the table. They are also the best combination of results. From the results of the table, it is apparent that the final score is much more sensitive to the number of estimators rather than the maximum number of features.

The cells marked “N/A” are those combinations of parameters for which the Kaggle scoreboard did not return a value. Unfortunately, I have no way of debugging the scoreboard, and the result files (included in the folder) look perfectly fine to me. Without the actual test labels, there is no way for me to generate the scores myself. However, I am including the parameters for completeness of the grid.

4 Results

4.1 Model Evaluation and Validation

As mentioned in the Methodology section, I tried a variety of Machine Learning models to approach the problem, and found that using Random Forests exceeded the benchmark (see Section

Estimators	Max. features	Score
10	16	0.81277
50	16	N/A
100	16	0.85975
10	32	0.81287
50	32	N/A
100	**32**	**0.86081**
10	100	0.79364
50	100	N/A
100	100	N/A

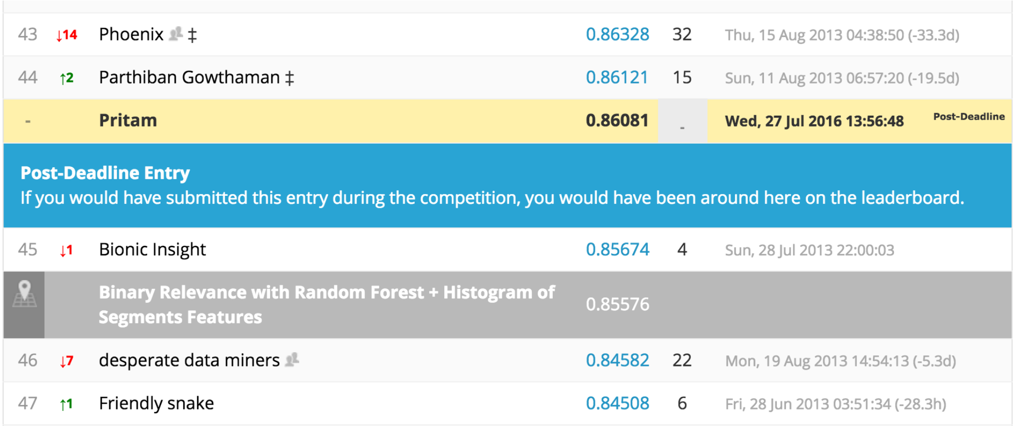
Table 3: Grid search results on the Random Forest Regressor

??). Further to that, using a manually performed grid search, I improved the performance of the algorithm (see Section **??**). A screenshot of the Kaggle results is shown in Figure **??**.

The split of training-vs-testing (50/50) and the high score on the competition leaderboard shows us that the code does generalise well to unseen data (assuming it is in a similar format). Note that the dataset already has the following issues in it, that have bean addressed while preprocessing:

1. Noise
2. Absence of any bird sounds
3. Multiple bird sounds
4. Overlapping bird sounds

4.2 Justification



43	↓14	Phoenix ‡	0.86328	32	Thu, 15 Aug 2013 04:38:50 (-33.3d)
44	↑2	Parthiban Gowthaman ‡	0.86121	15	Sun, 11 Aug 2013 06:57:20 (-19.5d)
-		**Pritam**	**0.86081**	-	**Wed, 27 Jul 2016 13:56:48** **Post-Deadline**

Post-Deadline Entry
If you would have submitted this entry during the competition, you would have been around here on the leaderboard.

45	↓1	Bionic Insight	0.85674	4	Sun, 28 Jul 2013 22:00:03
		Binary Relevance with Random Forest + Histogram of Segments Features	0.85576		
46	↓7	desperate data miners	0.84582	22	Mon, 19 Aug 2013 14:54:13 (-5.3d)
47	↑1	Friendly snake	0.84508	6	Fri, 28 Jun 2013 03:51:34 (-28.3h)

Figure 3: Best results obtained in this project

The benchmark score was **0.85576** and my score was **0.86081**, i.e. a small improvement over the original algorithm. It is, of course, significantly better than predicting the presence of birds at random. The analysis in Section **??** justifies my final choice of parameters.

5 Conclusion

5.1 Free-form visualisation

For the free-form visualization, I am including here a histogram of the number of species present in each recording is shown in Figure **??**. The histogram is particularly interesting because it shows the high percentage of recordings with **no birds** present in them.

Most of the recordings have only none or one species present in them. The high percentage of recordings with no birds is a result of these microphones being placed in the wild and autonomously recording throughout the day. Bird sounds are much more common in the morning and evening, and since the microphones were not properly optimised to account for this, a large number of the recordings have no birds in them.

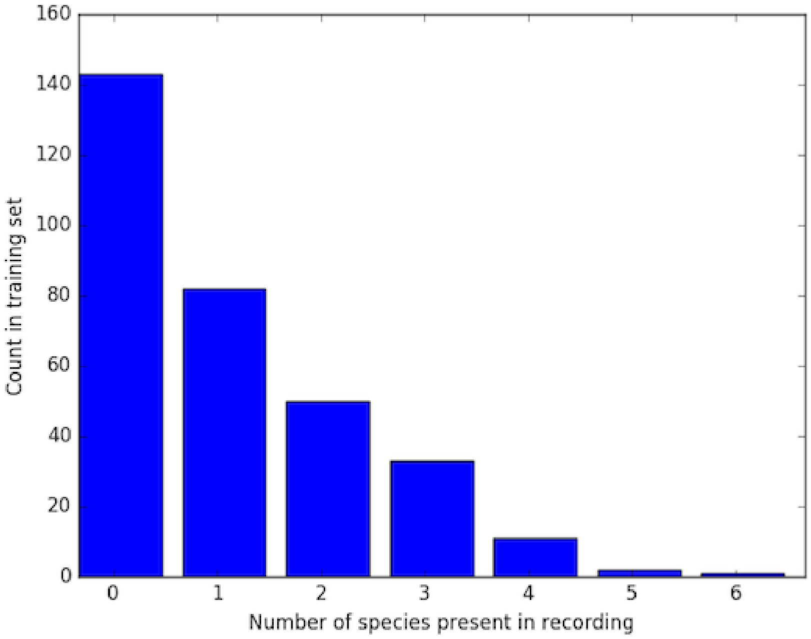


Figure 4: Histogram of number of birds in each recording

5.2 Reflection

The problem that I attempted to solve can be summarised as follows:

Given a recording (pre-shortened to 10 s), predict if there are any birds in the recording, and if so, how many and what species (of a subset of 20 species.

There are many challenges in going from audio to labelling of bird species, and through this project, I intended to understand these challenges and gain experience using the tools needed to overcome them as I move to a real-world setting. These challenges include:

1. Noise reduction

2. Separating overlapping bird sounds (the cocktail-party problem)

3. Localising birds sounds in the audio (segmentation)

4. Solving a Multi-Instance Multi-Label (MIML) problem

The final model as presented produced very impressive results. I am continuing to work on this topic and I hope to learn much more after finishing the MLND!

5.3 Improvement

One significant improvement I did not get around to implementing is including the location of the recording as an input. It is especially valid for this problem as the microphones are placed in fixed locations in the forests of Oregon. However, even in general, adding location as a variable is probably a very good idea since bird species are highly localised.

Another improvement that I do not yet have enough technical background for, is using Ensemble Classifier Chains (ECCs) instead of Binary Relevance for the MIML classifications. ECCs are known to perform much better than Binary Relevance for MIML problems, and in the future, I would love to learn more about ECCs and use them for classification.

References

[1] Forrest Briggs, Balaji Lakshminarayanan, Lawrence Neal, Xiaoli Z. Fern, and Raviv Raich *Acoustic classification of multiple simultaneous bird species: A multi-instance multi-label approach* 2012.

[2] Leo Breiman, Adele Cutler *Random Forests* `https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#intro`

[3] Decision tree `sklearn` documentation `http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`

[4] Random Forest Regressor `sklearn` documentation `http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html`

[5] Decision tree `sklearn` documentation `http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`

[6] Logistic Regression `sklearn` documentation `http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.htmll`

Appendix 1: Code Listings

Classification

```

from sklearn.ensemble import RandomForestRegressor

# Import functions in extract_data.py
```

```
from data_extract import extract_data

# Extract the data from the provided files
features_train, features_test, labels_train, test_ids = extract_data()

# The estimators to iterate over. This is used for both the manual
# grid search and to compare algorithms
ESTIMATORS = {
    # ”RFG”: RandomForestRegressor(n_estimators=100, max_features=32,
    #                                     random_state=0),
    ”RFG_1”: RandomForestRegressor(n_estimators=10, max_features=16,
                                   random_state=0),
    ”RFG_2”: RandomForestRegressor(n_estimators=50, max_features=16,
                                   random_state=0),
    ”RFG_3”: RandomForestRegressor(n_estimators=100, max_features=16,
                                   random_state=0),
    ”RFG_4”: RandomForestRegressor(n_estimators=10, max_features=32,
                                   random_state=0),
    ”RFG_5”: RandomForestRegressor(n_estimators=50, max_features=32,
                                   random_state=0),
    ”RFG_6”: RandomForestRegressor(n_estimators=100, max_features=32,
                                   random_state=0),
    ”RFG_7”: RandomForestRegressor(n_estimators=10, max_features=100,
                                   random_state=0),
    ”RFG_8”: RandomForestRegressor(n_estimators=50, max_features=100,
                                   random_state=0),
    ”RFG_9”: RandomForestRegressor(n_estimators=100, max_features=100,
                                   random_state=0),
    # ”K–nn”: KNeighborsRegressor(),
    # ”LR”: LinearRegression(),
    # ”Ridge”: RidgeCV(),
    # ”DT”: tree.DecisionTreeClassifier(),
}

### Perform the classification by iterating through the algorithms
### in ESTIMATORS()
y_test_predict = dict()
for name, estimator in ESTIMATORS.items():

    # Fit the model using the current algorithm
    estimator = estimator.fit(features_train, labels_train)
    # Predict
    y_test = estimator.predict(features_test)

    # Write the results to a named csv file based on the results
    # required by Kaggle
    with open('results' + name + '.csv', 'w') as results_file:
        results_file.write(”Id,Probability\n”)
        for i in range(len(test_ids)):
            for j in range(19):
                results_file.write(str(test_ids[i] * 100 + j) + ”,” +
                                   str(y_test[i][j]) + ”\n”)
```

Listing 1: Code for classification

Data extraction

```
import numpy as np
import pandas as pd

# This import is necessary to convert the MIML problem to a binary relevance
# problem
from sklearn.preprocessing import MultiLabelBinarizer


def extract_data():
    """
    Read the data files and extract data from them (training and test features,
    ids of test files, training labels
    :return: A list: ]features_train, features_test, labels_train, test_ids]
    """

    with open("essential_data/rec_labels_test_hidden.txt") as f:
        lines = f.readlines()

    labels_train = []
    training_ids = []
    test_ids = []
    for line in lines[1:]:
        if "?" not in line:
            x = map(int, line.split(','))
            labels_train.append(x[1:])
            training_ids.append(x[0])
        else:
            x = int(line.split(',')[0])
            test_ids.append(x)

    # Transform training_labels to nice format
    labels_train = MultiLabelBinarizer().fit_transform(labels_train)

    # Get the features from the files
    features_all = np.array(pd.read_csv
                            ('supplemental_data/histogram_of_segments.txt',
                             skiprows=1,
                             index_col=0,
                             header=None).values)
    features_train = features_all[training_ids]
    features_test = features_all[test_ids]

    # FOR DEBUGGING: Make sure sizes match
    print "Data size: " + str(len(lines[1:]))
    print "Feature set size: " + str(features_all.shape)
    print "Test size: " + str(len(test_ids))
    print "Training size: " + str(len(training_ids))
    print "Training feature set size: " + str(features_train.shape)
    print "Test feature set size: " + str(features_test.shape)
    return features_train, features_test, labels_train, test_ids
```

Listing 2: Code for data extraction from files