Machine Learning with Spotify dataset

Marcus Lee
marcustutorial@hotmail.com

November 22, 2023

1 Individual classifiers vs Ensemble (vote) of classifiers

1.1 Evaluation measure(s)

I chose **F1-Measure** because it is a combination of both precision and recall, hence it reflects both the true positive rate and false positive rate of the classifiers' performance which is why this measure depicts a more complete picture of each classifier's strengths and weaknesses compared to other measures such as accuracy. Hence, it is easier for us to compare and determine the trade-offs between classifiers.

To clarify, Weka labels it as `F-Measure` but it is the same as F1-Measure because it uses harmonic mean ($\beta = 1$) in the calculation of:

$$F = \frac{(1+\beta^2) \cdot \text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}$$

Note that in Weka, the weighted average of F-Measure was calculated by:

$$F_{\text{weighted avg}} = \frac{\sum_{k=1}^{n} F_k \cdot \text{count(k)}}{\sum_{k=1}^{n} \text{count(k)}}$$

where n is the number of classes, F_k is the F-Measure of class k and $\text{count}(k)$ is the number of instances of class k.

1.2 Evaluation of the three classifiers

The weka classifiers used for the evaluation are:

- **Decision Tree** → `weka.classifiers.trees.J48`
- **Neural Network** → `weka.classifiers.functions.MultiLayerPerceptron`
- **k-NN** → `weka.classifiers.lazy.IBk`

	Decision Tree	**Neural Network**	**k-NN (k=1)**
edm	0.593	0.608	0.542
latin	0.408	0.375	0.411
pop	0.343	0.360	0.331
rap	0.608	0.629	0.549
rock	0.653	0.679	0.608
Weighted Average	**0.522**	**0.532**	**0.488**

Table 1: F1-Measure of the three classifiers on the dataset

The table above shows the results of the classifiers, they were ran with default classifier settings and test options of 10-fold cross validation. In this case, k-fold cross validation was preferred over train test split so that the classifiers are trained on as much data as possible and because the dataset is small, the incurred computational overhead is acceptable.

As can be seen from the results, the *neural network* classifier performed the best with an average F1-Measure of **0.532**. The *decision tree* classifier performed the second best with an average F1-Measure of **0.522**. The *k-NN* classifier performed the worst with an average F1-Measure of **0.488**. One of the possible cause that neural network performed the best because it is capable of learning complex non-linear relationship across the features set.

1.3 Evaluation of the ensemble of the three classifiers

The combination rules used for the ensemble are:

- **Average of Probabilities**
- **Majority Voting**
- **Minimum Probability**

1.3.1 Results

To ensure the fairness of the test, the ensemble was also ran with 10-fold cross validation. The results for different combination rules are as follows:

	Average of Possibilities	**Majority Voting**	**Minimum Probability**
edm	0.614	0.621	0.540
latin	0.435	0.432	0.402
pop	0.366	0.378	0.345
rap	0.644	0.650	0.551
rock	0.696	0.703	0.618
Weighted Average	**0.552**	**0.557**	**0.491**

Table 2: F1-Measure of the ensemble of the three classifiers

Looking at the table above, we can see that *Majority Voting* produces the best result with an average F1-Measure of **0.557**. The *Average of Probabilities* produces the second best result with an average F1-Measure of **0.552**. The *Minimum Probability* produces the worst result with an average F1-Measure of **0.491**.

1.3.2 Results justification

Majority Voting performed the best because it utilises a simple voting scheme and since the dataset has discrete class labels such as `edm`, `latin`, `pop`, `rap` and `rock`, it is robust to noisy prediction wherein even a few classifiers produced incorrect predictions the influence on the final decision is minor as long as majority of the classifiers produced correct predictions.

As for *Minimum Probability*, it performed the worst due to its conservativeness in which it only focuses on the minimum probability. This is especially not ideal for this dataset because the dataset is small hence a lot of information was discarded when solely looking at the minimum probability.

The justification for *Average of Possibilities* is that although it does not perform as well as *Majority Voting*, it is a safe go-to approach since it factors in both the strengths and weaknesses of all classifiers by averaging their output.

In conclusion, comparing the overall results of the ensemble with the individual classifiers, we can see that the ensemble performed better than the individual classifiers. This is because the ensemble is able to combine the strengths of the individual classifiers and mitigate their weaknesses.

2 Ensemble with bagging

2.1 Results with increasing bag size

The configuration for the results are as follows:

- **Decision Tree** → `weka.classifiers.trees.J48 (10-fold cross validation)`
- **Neural Network** → `weka.classifiers.functions.MultiLayerPerceptron (80/20 train test split)`
- **k-NN** → `weka.classifiers.lazy.IBk (10-fold cross validation)`

The results below were ran using `weka.classifiers.meta.Bagging` with the above configuration and increasing `numIterations` (equivalent to bag size) from 2 to 20. Note that, neural network does not use 10-fold cross validation because it takes too long to run. For the sake of simplicity, only the results for *Weighted Average* is included.

	2	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**
Decision Tree (J48)	0.495	0.548	0.563	0.572	0.579	0.583	0.584	0.587	0.588	0.592
Neural Network	0.559	0.56	0.565	0.565	0.566	0.566	0.567	0.566	0.567	0.566
k-NN	0.467	0.484	0.489	0.492	0.492	0.494	0.494	0.494	0.495	0.495

Table 3: F1-Measure of the ensemble (bagging) with increasing bag size

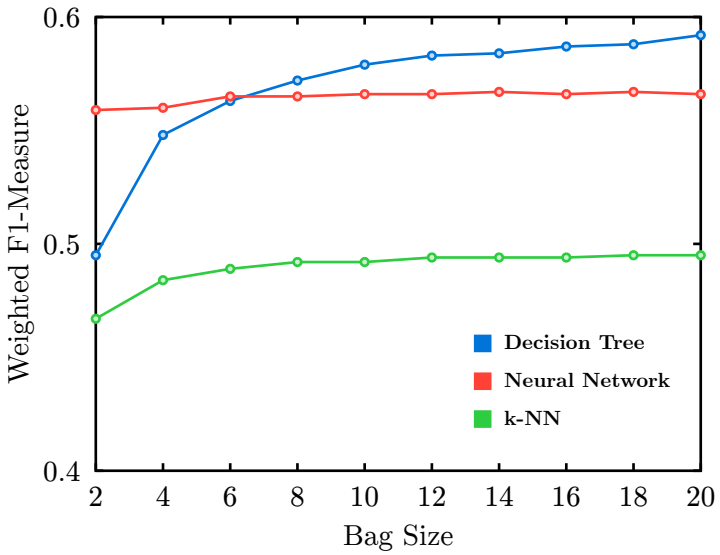

Figure 1: Plot of ensemble (bagging) with increasing bag size

From the table and figure above we can see a general trend for all three classifiers that as the bag size increases, the F1-Measure also increases. Hence the best performing ensemble size is the largest bag size, 20. This phenomenon can be explained by the *Condorcet Jury Theorem* where given the probability of each voter being correct is p and the probability of majority of voters being correct is M, then:

$$\text{if } p > 0.5, \text{then } M > p$$

$$\text{if } p \text{ always} > 0.5, \text{then } M \text{ approaches 1.0 as the number of voters approaches } \infty$$

In our case, as the bag size approaches infinity, the F1-measure of the ensemble approaches 1.0. However, the computation cost of the ensemble also increases linearly with the bag size. Hence, there is a trade-off between the computation cost and the performance of the ensemble so the advisable approach is to keep increasing the bag size until improvements are too tiny to be considered or running out of computation power.

Additionally, we can observe that for each classifier there seems to be a threshold bag size where increment after that have small impacts on the result. This is a phenomenon called "level-off" and will be discussed in the next section where the effect is more observable.

3 Ensemble with random subspacing

3.1 Results with increasing subspace size

The configuration for the results are as follows:

- **Decision Tree** → `weka.classifiers.trees.J48 (10-fold cross validation)`
- **Neural Network** → `weka.classifiers.functions.MultiLayerPerceptron (80/20 train/test split)`
- **k-NN** → `weka.classifiers.lazy.IBk (10-fold cross validation)`

The results below were ran using `weka.classifiers.meta.RandomSubSpace` with the above configuration and increasing `subSpaceSize` from 2 to 20. Note that, neural network does not use 10-fold cross validation because it takes too long to run. For the sake of simplicity, only the results for *Weighted Average* is included.

	2	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**
Decision Tree (J48)	0.482	0.527	0.556	0.572	0.573	0.522	0.522	0.522	0.522	0.522
Neural Network	0.451	0.507	0.533	0.559	0.558	0.557	0.557	0.557	0.567	0.567
k-NN	0.415	0.48	0.517	0.527	0.514	0.488	0.488	0.488	0.488	0.488

Table 4: F1-Measure of the ensemble (random subspacing) with increasing sub space size

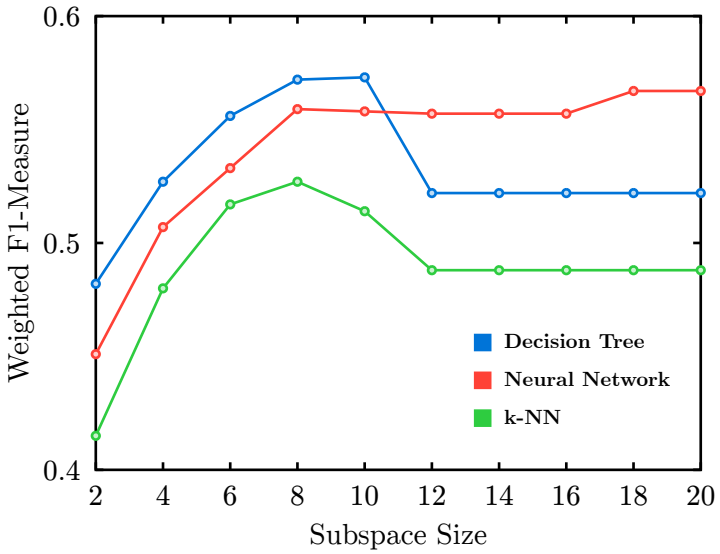

Figure 2: Plot of ensemble (random subspacing) with increasing sub space size

From the table and figure above, we see that from subspace size **2** to **8** all three classifiers shown an upward trend. However, after subspace size **8** the F1-Measure of all three classifiers started to level off and eventually plateus. This phenomenon is due to new ensemble members start to produce results similar to previous members hence no additional diversity are added causing the final output to be similar if not exact with the previous runs.

4 Suitable classfiers each ensemble method

4.1 Suitable classifiers for bagging ensemble

Based on the lectures, bagging is suitable for classifiers that are unstable. Unstable classifiers are classifiers that are sensitive to small change in the input data or in other words high variance. This is a sign that these classifiers are more prone to overfitting. Hence, bagging is suitable for unstable classifiers because it builds new models using the same classifier on variants of the data and if the classifier is very stable, it will just get similar results each time therefore not gaining much from the classifier. Several examples of unstable classifiers are decision trees and neural networks.

4.2 Suitable classifiers for random subspacing ensemble

On the other hand, random subspacing is suitable for classifiers that are stable. Stable classifiers are classifiers that are not sensitive to small change in the input data or in other words low variance. Hence, random subspacing is suitable for stable classifiers because it introduced randomness by randomly selecting a subset of features each time and by doing so made the base models more diverse and hence reducing model correlation. One such stable classifier are k-NN.

4.3 Best ensemble method for each classifer in this dataset

4.3.1 Decision Tree

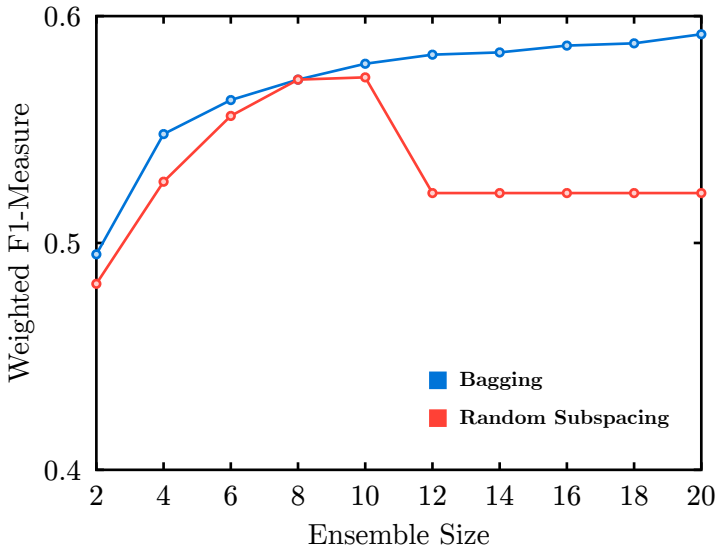

Figure 3: Performance of bagging ensemble against random subspacing ensemble

The figure above shows the performance of bagging ensemble against random subspacing ensemble for decision tree. As can be seen from the figure, bagging ensemble performed better than random subspacing ensemble. This is in line with expectation because decision tree is an unstable classifier and hence bagging ensemble is more suitable for it.

4.3.2 Neural Network

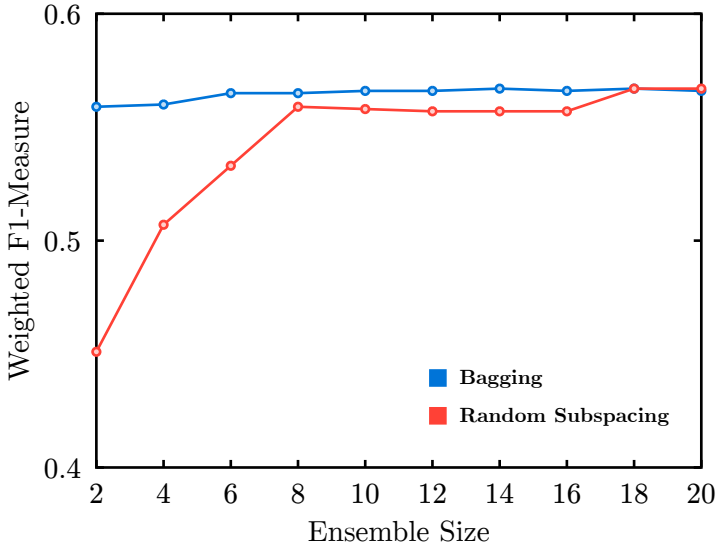

Figure 4: Performance of bagging ensemble against random subspacing ensemble

From the comparison figure above, we can see that bagging ensemble performed better than random subspacing ensemble. This is in line with expectation because neural network is an unstable classifier and bagging ensemble takes advantage of this characteristics.

4.3.3 k-NN

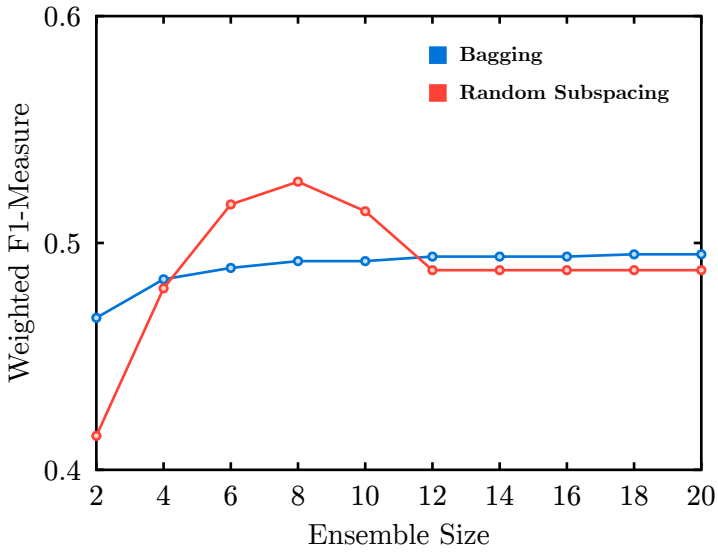

Figure 5: Performance of bagging ensemble against random subspacing ensemble

From the figure above, the performance for both strategy seems similar but if we average each measure, we would get **0.4896** for bagging and **0.4893** for random subspacing. Hence, on average bagging ensemble performed better than random subspacing ensemble although the gain is almost negligible. Interestingly, random subspace performed better than bagging for ensemble size from **6** to **10**. This is a little out of expectation since k-NN is a stable classifier, I would expect random subspace to outperform bagging. One possible cause might be because I am using $k = 1$ which might be too sensitive to noise and hence the ensemble is not able to mitigate the noise.

5 Linear Regression and Stochastic Gradient Descent

5.1 Results for Linear Regression

The source code for the linear regression can be found at `src/LinearRegression.java`. To run the code, use the following command:

```
make run-linear-regression
```

This will compile the code and run linear regression on the dataset. The model trained with 80% of the data and evaluated with 10-fold cross validation using 20% of the data, results are as follows:

Metrics	Value
Correlation coefficient	0.6922
Mean absolute error	0.0986
Root mean squared error	0.1248
Relative absolute error	71.2496%
Root relative squared error	72.1565%

Table 5: Evaluation metrics for linear regression

Looking at the result, it seems that there indeed is a linear relationship between the features `tempo`, `loudness` and `liveness` and the target variable `energy` as proven by the correlation coefficient of **0.6922** indicating it has a positive correlation. However, the model is not doing very well since the relative absolute error (RAE) is **71.2496%** and the root relative squared error (RRSE) is **72.1565%**. This is probably due to the fact that the dataset is small and hence the model is not able to learn the relationship between the features and the target variable well enough.

5.2 Results for Stochastic Gradient Descent

The source code for the linear regression can be found at `src/StochasticGradientDescent.java`. To run the code, use the following command:

```
make run-sgd
```

This will compile the code and run stochastic gradient descent with Squared Loss as the loss function on the dataset. The model trained with 80% of the data and evaluated with 10-fold cross validation using 20% of the data, results are as follows:

Metrics	Value
Correlation coefficient	0.6876
Mean absolute error	0.0992
Root mean squared error	0.1256
Relative absolute error	71.6696%
Root relative squared error	72.6127%

Table 6: Evaluation metrics for stochastic gradient descent

Surprisingly, the result for SGD is very similar to linear regression with its RAE and RRSE also in the 71% - 73% range. I would also consider that this model is also of poor quality because having a RAE of 71% means that the model is only able to predict the target variable correctly 29% of the time.

5.3 Differences between Linear Regression and Stochastic Gradient Descent

From the results above, we can see that the performance for both linear regression and stochastic gradient descent (SGD) is very similar with subtle differences. There are several possible reasons for this:

- **Similar Loss Function** - The loss function that the SGD model uses is Squared Loss and the linear regression by Weka might also be using a similar underlying loss function.
- **Dataset Size** - The dataset size is small and therefore both models might not be able to learn the relationship between the features and the target variable well enough to make a difference.
- **Linearity** - The linear relationship between the features and target variable might be too dominant and hence SGD is basically optimising the same problem as linear regression on a similar gradient line. One evidence to support this is that the correlation coefficient for both models are relatively moderate and similar to each other.