Concordia University

COMP6721

Introduction to Artificial Intelligence

Fall 2018

Mini-Project 2 Report

Author:
Adrien Poupa
40059458

Professor:
Dr. Leila Kosseim

November 14, 2018

Contents

1 Machine Learning in Python with Scikit

1.1 Structure of the Program

I chose to use the Python Scikit package because of its complete official documentation and the large amount of resources available online: tutorials, examples, etc. I was surprised by the simplicity of its use: only around 200 lines are enough to predict results with multiple classifiers.

I have created the following functions:

1. `train`: Train a given classifier, save the model with Joblib in a .joblib file and return the newly created classifier.

2. `load_classifier`: Load a classifier using a joblib file and return it.

3. `get_csv_data`: extract data from the CSV file while making sure to handle the specificities of the input file: no header, all but the last column should be returned as X since it is the data for every instance, the last column being the result should be returned as Y.

4. `get_best_parameters`: find the best hyperparameters given a classifier and a table of parameters to try. It uses `GridSearchCV` to do so.

5. `predict`: With a given trained classifier, call `get_csv_data` to extract data from the CSV file, predict the results, write them in a CSV output file, display accuracy when the expected values are known.

I called the `get_best_parameters`, `train` then the `predict` functions from the body of the script at the beginning to get the best hyperparameters and training the classifier that uses these parameters.

It takes a long time for the program to find the best hyperparameters for the six cases we have to cover: over an hour on a powerful i7 machine with 12GB of RAM. This is why after the first run, I use the `load_classifier` and `predict` functions to predict the results using a model that had already been trained.

The following references have been used while creating the script: [6] [16] [17] [5] [1] [3] [2].

1.2 Finding the Best Hyperparameters

Finding the best hyperparameters would be time-consuming, especially if one wants to combine them (ie: given the best hyperparameter A, which configuration should be used for hyperparameter B?). Fortunately, the Scikit package offers two grid searches models [10] that can be used to find the best hyperparameters. The following are the two brute force, general use search models that can be used on any classifier:

1. Exhaustive Grid Search (`GridSearchCV`): will exhaustively try all the combinations from an array containing all the parameters to search.

2. Randomized Parameter Optimization (`RandomizedSearchCV`): performs a random search over the given parameters, each setting being sampled from a distribution over possible parameter values.

Note that Scikit offers more search models other than brute force, such as `ElasticNetCV` or `MultiTaskElasticNetCV` but those are targeted towards specific classifiers. I chose to use `GridSearchCV` over `RandomizedSearchCV` because, while the run time is lower for the latter, the performance is better for the former [12]. This is not an issue for our use case since we will generate the models only once and store them with Joblib. Reusing them after generation removes all the overhead that comes with grid search.

The `get_best_parameters` function will take an array of hyperparameters (the `param_grid` argument) and a classifier to test against, and then return the best parameters it has found. Listing 1 shows how the parameters are found:

```
grid = GridSearchCV(estimator=classifier,
                    param_grid=param_grid, cv=5)
```

Code Listing 1: `GridSearchCV` usage in the `get_best_parameters` function

The `cv` parameter determines the cross-validation splitting strategy. It is set to 3 by default but setting it higher gives better results but higher run-time. I set it to 5 as the run-time is not much higher but the results are more accurate; moreover, this will become the new default setting for Scikit 0.22.

The `get_best_parameters` function will return the best hyperparameters for the train values, thus the best hyperparameters for this set may be different from those found for the val set [1].

2 Experimenting with Machine Learning Algorithms

This section will be presenting the classifiers used to predict the results, and the hyperparameters used to improve those results.

2.1 Bernoulli Naive Bayes

2.1.1 Available Classifiers

The Scikit package offers four classifiers based on the Naive Bayes algorithm: [9]

1. Gaussian Naive Bayes (`GaussianNB`): the probabilities of the features are assumed to be following a Gaussian pattern. It is not relevant to our dataset.

2. Multinomial Naive Bayes (`MultinomialNB`): uses a use a "bag of values" approach. For our dataset, it means that we would only care about the number of zeros and ones for each entry rather than their position. It is too simplistic for our dataset.

3. Complement Naive Bayes (`ComplementNB`): an adaptation of the Multinomial Naive Bayes that is targeted towards imbalanced data sets. Like the Multinomial Naive Bayes, it is too simplistic for our dataset.

4. Bernoulli Naive Bayes (`BernoulliNB`): the dataset may contain multiple features but each one is assumed to be a "binary-valued (Bernoulli, boolean) variable". This is what we have for the dataset, hence the choice to use this classifier [15].

2.1.2 Tweaking the Hyperparameters

Among the hyperparameters available for the `BernoulliNB` classifier [11], I have chosen the following:

1. `alpha`: float smoothing parameter. It is set to 1.0 by default. 0 means no smoothing. I chose to tweak this parameter because it is at the core of the model.

2. `fit_prior`: boolean flag, states if the model should learn class prior probabilities. Set to `True` by default. I chose to tweak this parameter because I wanted to see if a uniform prior had an impact on the result prediction or not.

To tweak those two hyperparameters, I used the code shown in listing 2.

```
# Hyperparameters for Naive Bayes
alphas = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5,
          0.6, 0.7, 0.8, 0.9, 1.0]
param_grid_nb = {'alpha': alphas, 'fit_prior': [True, False]}
```

Code Listing 2: Creating the array used to find the best hyperparameters

Then, the `param_grid_nb` array will be used to feed the `get_best_parameters` function and the `GridSearchCV` will train the model with all the alphas from 0.0 to 1.0 with both the `fit_prior` flag enabled or not. The best combination of hyperparameters will be returned; see listing 3.

```
best_parameters = get_best_parameters("1", BernoulliNB(),
                                      param_grid_nb)
classifier = train("1", "nb", BernoulliNB(**best_parameters))
```

Code Listing 3: Getting the best parameters and training the model

2.2 Decision Tree Classifier

2.2.1 Available Classifiers

The Scikit package offers two classifiers based on the Decision Tree algorithm: [7]

1. Decision Tree Classifier (`DecisionTreeClassifier`): a classifier that can do "multi-class classification on a dataset". This is what we will use for our dataset.

2. Decision Tree Regressor (`DecisionTreeRegressor`): used for regression problems, not relevant to the dataset.

2.2.2 Tweaking the Hyperparameters

Among the hyperparameters available for the `DecisionTreeClassifier`, [13], I have chosen the following:

1. `criterion`: function used to measure the quality of a split. The two possible options are "entropy" for the information gain and "gini" for the Gini impurity. It is set to "gini" by default. I chose to tweak this parameter because I wanted to see if the entropy that we studied in class was the best option.

2. `splitter`: defines how the split at each node should be selected. Available options are "best" by default to take the best split or "random' to choose a best random split. I chose to tweak this parameter because I wanted to see the impact of the splitting technique for decision trees.

3. `presort`: boolean flag, states if the data should be presort to accelerate the finding of best splits while fitting. It is set to `False` by default. I chose to tweak this parameter because I wanted to see if presorting had an impact on the result prediction or not.

To tweak those three hyperparameters, I used an array similar to code shown in listing 2. Then, the `param_grid_dt` array will be used to feed the `get_best_parameters` function and the `GridSearchCV` will train the model with the `criterion` set to Gini or Entropy, the `splitter` to Best and Random, and the `presort` set to `True` or `False`. The best combination of hyperparameters will be returned.

2.3 Multi-layer Perceptron

2.3.1 Available Classifiers

The Scikit package offers two classifiers based on the Neural network algorithm: [8]

1. Multi-layer Perceptron (`MLPClassifier`): a classifier that trains using backpropagation, similarly to what was studied in class. I have chosen this classifier because I wanted to see to compare the results among the three algorithms presented in class for a real-life dataset.

2. Multi-layer Perceptron Regressor (`MLPRegressor`): about the same as the `MLPClassifier` except that it does not use an activation function but the "the square error as the loss function".

2.3.2 Tweaking the Hyperparameters

Among the hyperparameters available for the `MLPClassifier` [14], I have chosen the following:

1. `activation`: defines the activation function for the hidden layer. Four activation functions are available:

   (a) `identity`: the function $f(x) = x$.

   (b) `logistic`: the function $f(x) = \frac{1}{1+\exp(-x)}$.

   (c) `tanh`: the function $f(x) = \tanh(x)$.

   (d) `relu`: the function $f(x) = \max(0, x)$.

   I chose to tweak this parameter because I wanted to see if the `logistic` function that we used in class was the most efficient.

2. `solver`: defines the optimization algorithm. Three algorithms are available:

   (a) `lbfgs`: stands for L-BFGS, the L meaning "Limited-memory". It is technique based on the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm, belonging to quasi-Newton methods, finding zeroes or local maxima and minima of functions. L-BFGS approximates BFGS. [18] [19] [20]

   (b) `sgb`: acronym for "stochastic gradient descent".

   (c) `adam`: the ADAM algorithm as described in [4].

   I chose to tweak this parameter because I wanted to see which solving algorithm was the most efficient.

3. `learning_rate`: defines how the learning rate should evolve. Three alternatives are available:

   (a) `constant`: use a constant learning rate defined by the `learning_-rate_init` parameter (set to 0.001 by default).

   (b) `invscaling`: gradually decrease the learning rate.

   (c) `adaptive`: adapt the learning rate based on the `tol` (tolerance) parameter that is set to 0.0001 by default. Keep the learning rate constant to `learning_rate_init` if training loss decreases. Whenever two consecutive epochs fail to decrease training loss by `learning_rate_init` or more, divide the current learning rate by 5.

   This is only used when the solver is set to `sgb`. I chose to tweak this parameter because I wanted to see if a constant learning rate was good enough for large datasets or if other methods were preferable.

To tweak those three hyperparameters, I used an array similar to code shown in listing 2. Then, the `param_grid_mlp` array will be used to feed the `get_-best_parameters` function and the `GridSearchCV` will train the model with the `activation` set to `identity`, `logistic`, `tanh` and `relu`; `solver` will be set to to `lbfgs`, `sgb` and `adam`; `learning_rate` will be set to to `constant`, `invscaling` and `adaptive`. The best combination of hyperparameters will be returned.

3 Result Analysis

3.1 Dataset 1

3.1.1 Final Results

Table 1 gives the results for the first dataset.
"BH" stands for "Best Hyperparameters".

Algorithm	**Base Accuracy**	**BH Accuracy**	**BH**
Bernoulli Naive Bayes	59.92%	60.12%	`alpha`: 0.1 `fit_prior`: False
Decision Tree	27.63%	29.77%	`criterion`: gini `presort`: False `splitter`: best
Multi-layer Perceptron	63.42%	64.98%	`activation`: logistic `learning_rate`: invscaling `solver`: adam

Table 1: Results for the first dataset

One can see that using a Decision Tree algorithm for this dataset gives the worst results by far. Bernoulli Naive Bayes and Multi-layer Perceptron give better results, especially for the latter.

Using a grid search to find the best hyperparameters improve the accuracy between the train set and the val set by 0.2% for Bernoulli Naive Bayes, 2.14% for Decision Tree and 1.56% for Multi-later Perceptron. All in all, this is negligible.

Even for the most efficient algorithm of the three, Multi-layer Perceptron, the performance is not good, meaning that just over one instance out of two will be correctly predicted.

The accuracy is low because the training set is not large enough. As shown in figure 1, only the first 10 characters have more than 50 occurrences in the training set, and the most common (0th character, hence the "A") is only repeated 238 times.

Thus, even though there is a correlation between the training set and the val set as highlighted by the figure 1, there simply is not enough data, at least for characters 10 to 50, to be correctly predicted by our machine learning models.

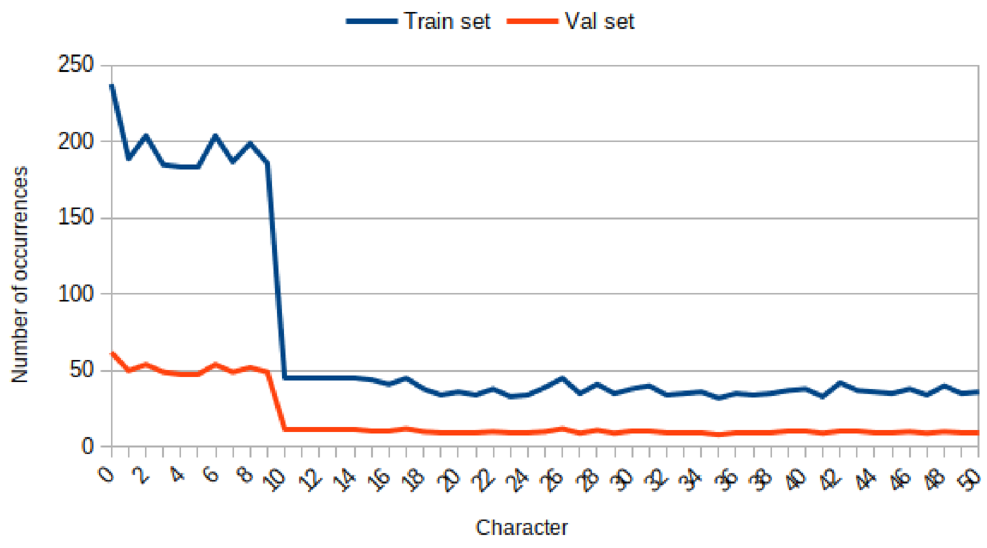

Figure 1: Correlation between the train set and the val set for dataset 1

3.1.2 Experimenting With the Bernoulli Naive Bayes Algorithm

First, we will manually tweak the `alpha` hyperparameter to see its influence on accuracy for the first set. The `alpha` hyperparameter ranges from 0 to 1. The results are shown in table 2.

Alpha	0	0.1	0.2	0.3	0.4
Accuracy for train set	58.67%	**60.61%**	59.85%	59.44%	59.03%
Accuracy for val set	58.56%	60.12%	60.70%	60.51%	60.70%

Alpha	0.5	0.6	0.7	0.8	0.9	1
Accuracy for train set	58.83%	58.37%	58.16%	57.65%	57.40%	56.99%
Accuracy for val set	**61.09%**	60.89%	60.51%	60.12%	60.12%	59.92%

Table 2: Tweaking the `alpha` parameter for the `BernoulliNB` classifier

Here, one can see that the grid search returned the value of 0.1 as best `alpha` since it is the one that performs best for the train set. Alas, on the val set, it is 0.5 that would have given the best results. For the val set, there seems to be a Gaussian-like curve for the accuracy peaking at 0.5.

For the `fit_prior` hyperparameter, table 3 shows the results for the first dataset.

Fit prior	True	False
Accuracy for train set	56.99%	56.99%
Accuracy for val set	59.92%	59.73%

Table 3: Tweaking the `fit_prior` parameter for the `BernoulliNB` classifier

Interestingly, the `fit_prior` does not impact the accuracy for the test set and only changes the accuracy between the train set and the val set by 0.19%; thus, is is negligible.

3.1.3 Experimenting With the Decision Tree Algorithm

Table 4 presents the accuracy results after tweaking the `criterion`.

Criterion	Gini	Entropy
Accuracy for train set	28.21%	27.91%
Accuracy for val set	26.85%	30.35%

Table 4: Tweaking `criterion` for the `DecisionTreeClassifier`

Both Gini and Entropy have a similar accuracy for the test set, leading the grid search to select Gini, slightly better than Entropy by 0.3%, even though Entropy would have performed better on the val set by 3.5%.

Table 5 presents the accuracy results after tweaking the `splitter` hyperparameter.

Splitter	Best	Random
Accuracy for train set	28.21%	28.32%
Accuracy for val set	26.85%	27.24%

Table 5: Tweaking `splitter` for the `DecisionTreeClassifier`

Best and Random give similar results for the test set, Random being 0.11% more efficient. This leads to a prediction for the val set that is 0.39% more accurate.

Table 6 presents the accuracy results after tweaking the `presort` hyperparameter.

Presort	True	False
Accuracy for train set	28.21%	28.21%
Accuracy for val set	26.85%	26.85%

Table 6: Tweaking `presort` for the `DecisionTreeClassifier`

Interestingly, this hyperparameter has no influence at all on the prediction, whether this is for the test set or the val set.

3.1.4 Experimenting With the Multi-layer Perceptron Algorithm

Table 7 presents the accuracy results after tweaking the `activation` hyperparameter.

Activation	Identity	Logistic	Tanh	Relu
Accuracy for train set	61.84%	65.15%	63.16%	62.55%
Accuracy for val set	60.31%	66.34%	63.04%	63.62%

Table 7: Tweaking the `activation` parameter for the `MLPClassifier`

The logistic function performs better than the others by far, this is true for both the test set and the val set.

Table 8 presents the accuracy results after tweaking the `solver` hyperparameter.

Solver	LBFGS	SGD	Adam
Accuracy for train set	57.86%	56.38%	62.55%
Accuracy for val set	62.65%	58.37%	63.62%

Table 8: Tweaking the `solver` parameter for the `MLPClassifier`

The Adam solver performs better than the others by far, this is true for both the test set and the val set. However, this difference is more noticeable on the test than the val set. Interestingly, the LBFGS solver is 4.79% more efficient on the test set than the val set.

Table 9 presents the accuracy results after tweaking the `learning_rate` hyperparameter.

Learning rate	Constant	Invscaling	Adaptive
Accuracy for train set	62.55%	62.55%	62.55%
Accuracy for val set	63.62%	63.62%	63.62%

Table 9: Tweaking the `learning_rate` parameter for the `MLPClassifier`

This hyperparameter has no influence at all on the prediction, whether this is for the test set or the val set.

3.2 Dataset 2

3.2.1 Final Results

Table 10 gives the results for the second dataset.
"BH" stands for "Best Hyperparameters".

Algorithm	**Base Accuracy**	**BH Accuracy**	**BH**
Bernoulli Naive Bayes	80.05%	80.35%	`alpha`: 0.0 `fit_prior`: True
Decision Tree	76.95%	77.05%	`criterion`: gini `presort`: False `splitter`: random
Multi-layer Perceptron	88.40%	90.55%	`activation`: logistic `learning_rate`: constant `solver`: adam

Table 10: Results for the second dataset

There is a clear difference in terms of accuracy for this dataset. The Multi-layer Perptron with default hyperparameters has an accuracy that is 11.45% better than default Decision Tree. Once again, tweaking the hyperparameters gives better results but the difference is not significant.

Compared to the first dataset, it seems possible to predict results for this dataset in a confident manner; a 90.55% accuracy for the Multi-layer Perceptron with the best hyperparameters is good enough so the system could be used.

It is most likely easier to predict results for this dataset because it only has 10 different output compared to 50 for the first dataset. There are no duplicate in the training data.

Another reason why the accuracy is high for this dataset is the tight relationship between the training set and the val set. As table 11 shows, there is a strong correlation in terms of percentage of occurrences between those two sets. Moreover, there are at least 200 occurrences and up to 1500 occurrences for each output in the training set: this is enough data to feed our models.

Index	**Character**	**Occurrences in the train set**	**%**	**Occurrences in the val set**	**%**
0	pi	700	10.9375	200	10
1	alpha	1500	23.4375	500	25
2	beta	200	3.125	50	2.5
3	sigma	200	3.125	50	2.5
4	gamma	600	9.375	200	10
5	delta	700	10.9375	200	10
6	lambda	200	3.125	50	2.5
7	omega	200	3.125	50	2.5
8	mu	600	9.375	200	10
9	xi	1500	23.4375	500	25
Total		6400	100	2000	100

Table 11: Correlation between the train set and the val set for dataset 2

One can see the strong correlation in terms of percentage for the occurrences between the two sets. It is even more obvious in figure 2.

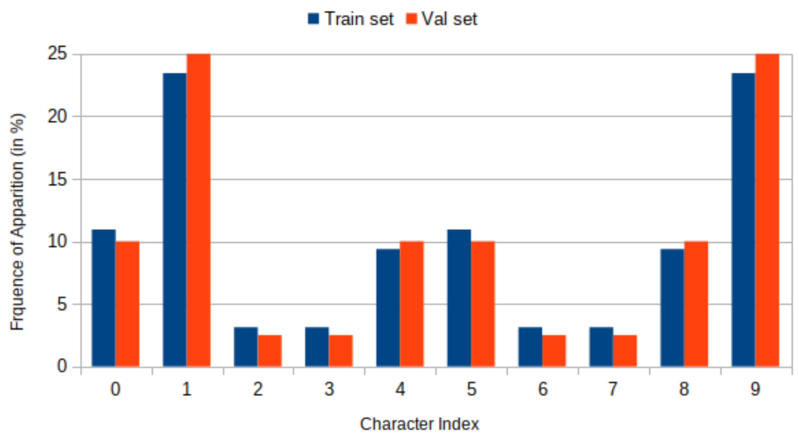

Figure 2: Correlation between the train set and the val set for dataset 2

3.2.2 Experimenting With the Bernoulli Naive Bayes Algorithm

We will manually tweak the `alpha` hyperparameter to see its influence on accuracy for the second set. The `alpha` hyperparameter ranges from 0 to 1. The results are shown in table 12.

Alpha	0	0.1	0.2	0.3	0.4
Accuracy for train set	**80.67%**	80.50%	80.42%	80.41%	80.30%
Accuracy for val set	**80.35%**	80.30%	80.20%	80.15%	80.20%

Alpha	0.5	0.6	0.7	0.8	0.9	1
Accuracy for train set	80.27%	80.23%	80.19%	80.19%	80.11%	80.05%
Accuracy for val set	80.25%	80.10%	80.10%	80.05%	80.10%	80.05%

Table 12: Tweaking the `alpha` parameter for the `BernoulliNB` classifier

The grid search will select an `alpha` of 0. Unlike in the first dataset, the best `alpha` value for the test set also gives the best accuracy for the val set.

Table 13 presents the accuracy results after tweaking the `fit_prior` hyperparameter.

Fit prior	True	False
Accuracy for train set	80.05%	79.92%
Accuracy for val set	80.05%	79.90%

Table 13: Tweaking the `fit_prior` parameter for the `BernoulliNB` classifier

There is no significant difference in terms of accuracy for the `fit_prior` parameter; the `True` value will be selected because it is 0.13% more accurate for the test set.

3.2.3 Experimenting With the Decision Tree Algorithm

Table 14 presents the accuracy results after tweaking the `criterion`.

Criterion	Gini	Entropy
Accuracy for train set	74.59%	73.83%
Accuracy for val set	76.75%	75.95%

Table 14: Tweaking `criterion` for the `DecisionTreeClassifier`

The Gini criterion will be used because it is 0.76% more accurate than Entropy for the test set. The difference is negligible.

Table 15 presents the accuracy results after tweaking the `splitter` hyperparameter.

Splitter	Best	Random
Accuracy for train set	74.59%	74.28%
Accuracy for val set	76.75%	76.15%

Table 15: Tweaking `splitter` for the `DecisionTreeClassifier`

The Best splitter will be used because it is 0.31% more accurate than Random for the test set. The difference is negligible.

Table 16 presents the accuracy results after tweaking the `presort` hyperparameter.

Presort	True	False
Accuracy for train set	74.59%	74.59%
Accuracy for val set	76.75%	76.75%

Table 16: Tweaking `presort` for the `DecisionTreeClassifier`

This hyperparameter has no influence at all on the prediction, whether this is for the test set or the val set.

3.2.4 Experimenting With the Multi-layer Perceptron Algorithm

Table 17 presents the accuracy results after tweaking the `activation` hyperparameter.

Activation	Identity	Logistic	Tanh	Relu
Accuracy for train set	82.55%	89.00%	89.09%	86.47%
Accuracy for val set	83.35%	90.15%	89.65%	88.35%

Table 17: Tweaking the `activation` parameter for the `MLPClassifier`

Logistic and Tanh perform better than Identity and Relu. From this table alone, Tanh is the most performant but the grid search will pick Logistic because it performs better paired with the Adam solver.

Table 18 presents the accuracy results after tweaking the `solver` hyperparameter.

Solver	LBFGS	SGD	Adam
Accuracy for train set	83.80%	86.95%	86.47%
Accuracy for val set	85.55%	86.75%	88.35%

Table 18: Tweaking the `solver` parameter for the `MLPClassifier`

The SGD is the best solver in terms of accuracy for the test set, however the Adam solver is the best for the val set.

Table 19 presents the accuracy results after tweaking the `learning_rate` hyperparameter.

Learning rate	Constant	Invscaling	Adaptive
Accuracy for train set	86.47%	86.47%	86.47%
Accuracy for val set	88.35%	88.35%	88.35%

Table 19: Tweaking the `learning_rate` parameter for the `MLPClassifier`

This hyperparameter has no influence at all on the prediction, whether this is for the test set or the val set.

4 Conclusion

The two datasets are drastically different. It was not possible to find a reliable model to predict values for the first one (64.98% of accuracy for the best algorithm) whereas the second dataset peaked to 90.55%.

The difference of accuracy between the two models can be explained by looking at the training data. For the first dataset, training data is lacking in terms of quantity: only 10 characters are repeated at least 186 times. Others are only defined from 36 to 45 times; this is not enough for our models. On the other hand, the second dataset, much simpler with only 10 different outputs compared to 50, is much more predictable: its training set contain each ouput from 200 to 1500 times.

At the beginning of the mini-project, I had not realized the importance of the training set. It became obvious when I wanted to understand why the best hyperparameters for the training set would give results not as good as expected for the value set.

Both times, the Multi-layer Perceptron algorithm proved to be the most reliable. I expected the Bernoulli Naive Bayes algorithm to be the less accurate given its simplicity, but the Decision Tree algorithm turned out to perform worse. The Multi-layer Perceptron algorithm took longer to execute but gave better results than the other ones.

I was surprised to see how little the hyperparameters influenced the accuracy of the prediction. I expected a 5 to 10% difference, but it was a 1 to 2% increase in most of the cases.

5 Future Work

Future work could include testing other hyperparameters such as the `learning_rate_init` for the `MLPClassifier` that specifies the initial value for the learning rate.

Other classifiers could be tested, such as the Unsupervised Nearest Neighbors algorithm, or the Gaussian Process Classification.

References

[1] Brendan Barnwell. *Why does not GridSearchCV give best score ? - Scikit Learn*. 2018. URL: https://stackoverflow.com/questions/30442259/why-does-not-gridsearchcv-give-best-score-scikit-learn.

[2] Jason Brownlee. *How to Tune Algorithm Parameters with Scikit-Learn*. 2018. URL: https://machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-learn/.

[3] BrunoGL. *How to adjust the hyperparameters of MLP classifier to get more perfect performance*. 2018. URL: https://datascience.stackexchange.com/questions/36049/how-to-adjust-the-hyperparameters-of-mlp-classifier-to-get-more-perfect-performa.

[4] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

[5] Vivek Kumar. *RandomizedSearchCV best params changes everytime when I run this program*. 2018. URL: https://stackoverflow.com/a/49146736/10017187.

[6] Martin Müller. *Naive Bayes Classification With Sklearn*. 2018. URL: https://blog.sicara.com/naive-bayes-classifier-sklearn-python-example-tips-42d100429e44.

[7] Scikit. *1.10. Decision Trees - scikit-learn 0.20.0 documentation*. 2018. URL: https://scikit-learn.org/stable/modules/tree.html.

[8] Scikit. *1.17 Neural network models (supervised) - scikit-learn 0.20.0 documentation*. 2018. URL: https://scikit-learn.org/stable/modules/neural_networks_supervised.html.

[9] Scikit. *1.9. Naive Bayes - scikit-learn 0.20.0 documentation*. 2018. URL: https://scikit-learn.org/stable/modules/naive_bayes.html.

[10] Scikit. *3.2. Tuning the hyper-parameters of an estimator*. 2018. URL: https://scikit-learn.org/stable/modules/grid_search.html.

[11] Scikit. *BernoulliNB*. 2018. URL: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html.

[12] Scikit. *Comparing randomized search and grid search for hyperparameter estimation*. 2018. URL: https://scikit-learn.org/stable/auto_examples/model_selection/plot_randomized_search.html#sphx-glr-auto-examples-model-selection-plot-randomized-search-py.

[13] Scikit. *DecisionTreeClassifier*. 2018. URL: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

[14] Scikit. *MLPClassifier*. 2018. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html.

[15] StackOverflow. *Bernoulli NB vs MultiNomial NB, How to choose among different NB algorithms?* 2018. URL: https://stats.stackexchange.com/questions/258458/bernoulli-nb-vs-multinomial-nb-how-to-choose-among-different-nb-algorithms.

[16] Packt Editorial Staff. *Implementing 3 Naive Bayes classifiers in scikit-learn*. 2018. URL: https://hub.packtpub.com/implementing-3-naive-bayes-classifiers-in-scikit-learn/.

[17] Josh Terrell. *Eliminating warnings from scikit-learn*. 2018. URL: https://stackoverflow.com/a/33616192.

[18] Wikipedia. *Broyden–Fletcher–Goldfarb–Shanno algorithm*. 2018. URL: https://en.wikipedia.org/wiki/Broyden_Fletcher_Goldfarb_Shanno_algorithm.

[19] Wikipedia. *Limited-memory BFGS*. 2018. URL: https://en.wikipedia.org/wiki/Limited-memory_BFGS.

[20] Wikipedia. *Quasi-Newton method*. 2018. URL: https://en.wikipedia.org/wiki/Quasi-Newton_method.

A Expectations of Originality

Faculty of Engineering and Computer Science
Expectations of Originality

This form sets out the requirements for originality for work submitted by students in the Faculty of Engineering and Computer Science. Submissions such as assignments, lab reports, project reports, computer programs and take-home exams must conform to the requirements stated on this form and to the Academic Code of Conduct. The course outline may stipulate additional requirements for the course.

1. Your submissions must be your own original work. Group submissions must be the original work of the students in the group.
2. Direct quotations must not exceed 5% of the content of a report, must be enclosed in quotation marks, and must be attributed to the source by a numerical reference citation[1]. Note that engineering reports rarely contain direct quotations.
3. Material paraphrased or taken from a source must be attributed to the source by a numerical reference citation.
4. Text that is inserted from a web site must be enclosed in quotation marks and attributed to the web site by numerical reference citation.
5. Drawings, diagrams, photos, maps or other visual material taken from a source must be attributed to that source by a numerical reference citation.
6. No part of any assignment, lab report or project report submitted for this course can be submitted for any other course.
7. In preparing your submissions, the work of other past or present students cannot be consulted, used, copied, paraphrased or relied upon in any manner whatsoever.
8. Your submissions must consist entirely of your own or your group's ideas, observations, calculations, information and conclusions, except for statements attributed to sources by numerical citation.
9. Your submissions cannot be edited or revised by any other student.
10. For lab reports, the data must be obtained from your own or your lab group's experimental work.
11. For software, the code must be composed by you or by the group submitting the work, except for code that is attributed to its sources by numerical reference.

You must write one of the following statements on each piece of work that you submit:
For individual work: **"I certify that this submission is my original work and meets the Faculty's Expectations of Originality",** with your signature, I.D. #, and the date.
For group work: **"We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality",** with the signatures and I.D. #s of all the team members and the date.

A signed copy of this form must be submitted to the instructor at the beginning of the semester in each course.

I certify that I have read the requirements set out on this form, and that I am aware of these requirements. I certify that all the work I will submit for this course will comply with these requirements and with additional requirements stated in the course outline.

Course Number:	COMP6721	Instructor:	Dr Leila Kosseim
Name:	Adrien Poupa	I.D. #	40059458
Signature:	[illegible]	Date:	October, 5th 2018

[1] Rules for reference citation can be found in "Form and Style" by Patrich MacDonagh and Jack Bordan, fourth edition, May, 2000, available at http://www.encs.concordia.ca/scs/Forms/Form&Style.pdf.
Approved by the ENCS Faculty Council February 10, 2012