Kaggle Competition IFT3395/6390

Team: **Mokarian**

Ryan Mokarian, student #:20167536

Maysam Mokarian, student #:20167255

Introduction

This is a text classification project to take part in a Kaggle competition. The dataset are posts from Reddit and the goal is to use three algorithms of machine learning for topic identification of Reddit's posts.

To prepare our data, we used Nltk library for removing stop words and using stemming and tfidf. For the first milestone, we used a Naive Bayes classifier using Bag of words features and with additional Laplace smoothing. For the second milestone, we tested several models and we selected Logistic Regression and Multi Layer Perceptron. Results of the three models summarized below.

Model	Naive Bayes	Logistic Regression	MultiLayer Perceptron
Accuracy	0.56	0.54	0.59

Feature Design

Our pre-processing started with tokenizing the posts so that long texts to be splitted into smaller pieces/words, called token. Our features are words presented in the posts. Some words such as stop words [1] as well as punctuation and etc do not add any values in the process of training and prediction. To remove these type of features from out datasets, we decided to use several common pre-processing methods utilizing scikit learn libraries. Details explained below.

Tokenization: We have used nltk built in function (nltk.tokenize) [2] to tokenize the posts both for test and training data.

Asciifolding: We changed all the strings in the post to lowercase to avoid distinction between similar tokens with different asciifolding representation.

Removing stop words: We used two methods to remove the stopwords. First we calculated the 0.5% of the most frequent words and we considered them as stop words. After running k-fold cross validation, we noticed using nltk built in stopwords leads to a better performance, Hence we used the built in nltk.stopwords stopwords.words("english")[3].

Lemmatization and Stemming: we tried both lemmatization and stemming to transform the tokens in our vocabulary to their roots. After running two separate experiments, we noticed not using them resulted in higher accuracy and therefore better validation score.

TFIDF: we used TF-IDF (frequency-inverse document frequency) [4] to measure the importance of a token in a document in our dataset. As shown in below formula, importance of a word increases when the word used more frequently in a document; but it is inversely proportional to the number of times that the word is in the corpus.

$$TF - IDF(t, d) = \frac{count(t, d)}{|d|} \cdot \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

TF-IDF formula [4]

We used Feature_extraction [5] library from scikit learn that contains modules such as CountVectorizer, TfidfTransformer and TfidfVectorizer. As part of pre-processing we applied them on our dataset.

Algorithms

For each of our algorithms we used a pipeline which first went through pre-processing of data, count Vectorizer and TFIDFTransformer and then fit the processed data to a selected model. In the following, a review from each selected algorithms are presented.

Naive Bayes

Our first model is a generative model, Multinomial Naive Bayes classifier. The algorithm uses a Multinomial distribution and it is suitable for classification with discrete features, i.e. word counts for text classification of this project.
We tuned the alpha hyperparameter and we noticed the best accuracy when alpha=0.25

Logistic Regression

As opposed to our first generative model, for the second algorithm we used a discriminative model, Logistic Regression. This model predicts one of the trained 20 classes for a given piece of text (a testing post). We used the model for penalty l1 and l2 and obtained a better accuracy value for the l2 (l1: 0.52, l2: 0.54).

Multilayer Perceptron

The third used algorithm is Multilayer Perceptron [7], which is a class of feed forward artificial neural network. We selected this model as it is widely used currently in many applications and unlike Naive Bayes Classifier relies on an underlying Neural Network to perform the task of text classification. We used 100 hidden layer, relu as the activation function and adam solver. This algorithm provided us with best accuracy compared to the other two algorithms (0.59).

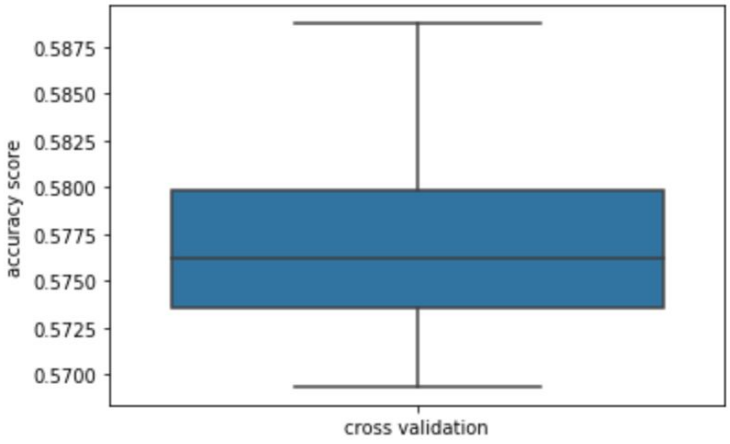
Methodology

In order to evaluate our models in general and to have an idea about the prediction performance of our models on new data we used k fold cross-validation. For the number of folds, we used 10 as a value of k=10 is very common in the field of applied machine learning.

Naive Bayes

10 fold cross-validation results on the accuracy scores for Naive Bayes algorithm is illustrated below. Low value of the standard deviation, 0.006, indicates that there is a high level of confidence on the accuracy score with mean value of 0.58.

mean	0.577275
std	0.005638
min	0.569340
25%	0.573559
50%	0.576216
75%	0.579799
max	0.588738

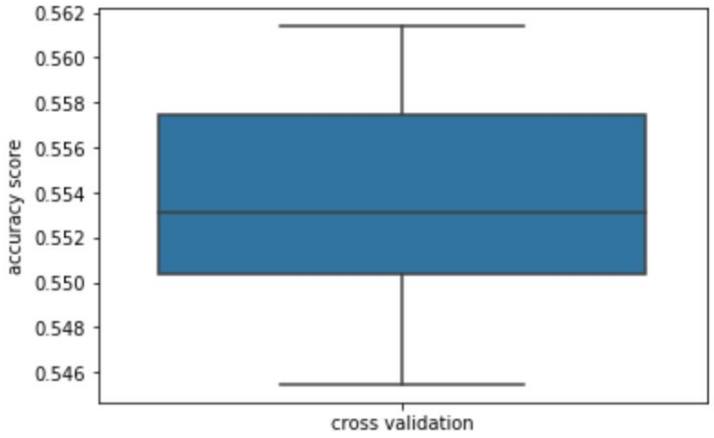


Result of 10-fold cross validation for Multinomial Naive Bayes classification

Logistic Regression

10 fold cross-validation results on the accuracy scores for Logistic Regression algorithm is illustrated below. Low value of the standard deviation, 0.005, indicates that there is a high level of confidence on the accuracy score with mean value of 0.55.

mean	0.553390
std	0.005284
min	0.545455
25%	0.550384
50%	0.553095
75%	0.557490
max	0.561378

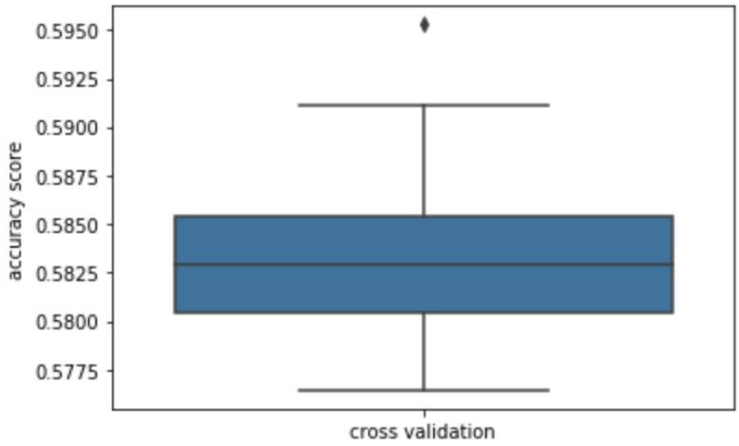


Result of 10-fold cross validation forLogistic regression

Multilayer Perceptron

10 fold cross-validation results on the accuracy scores for Multilayer Perceptron algorithm is illustrated below. Low value of the standard deviation, 0.006, indicates that there is a high level of confidence on the accuracy score with mean value of 0.58.

mean	0.583800
std	0.005870
min	0.576429
25%	0.580464
50%	0.582929
75%	0.585393
max	0.595286



Result of 10-fold cross validation for MLP

Results

Hyper parameters optimization: We used exhaustive search to tune our hyperparameters. GridSearchCV [5] from scikit learn is being used to suggest the best combination of some of the selected hyper parameters for our three used models. Additionally after trying two possible states of smooth_idf TfIdfTransformer, we noticed that setting this flag to False results in a better accuracy.

Naive Bayes

Setting hyper parameters: Using exhaustive search of GridSearchCV we tried all different combinations of the following parameters:

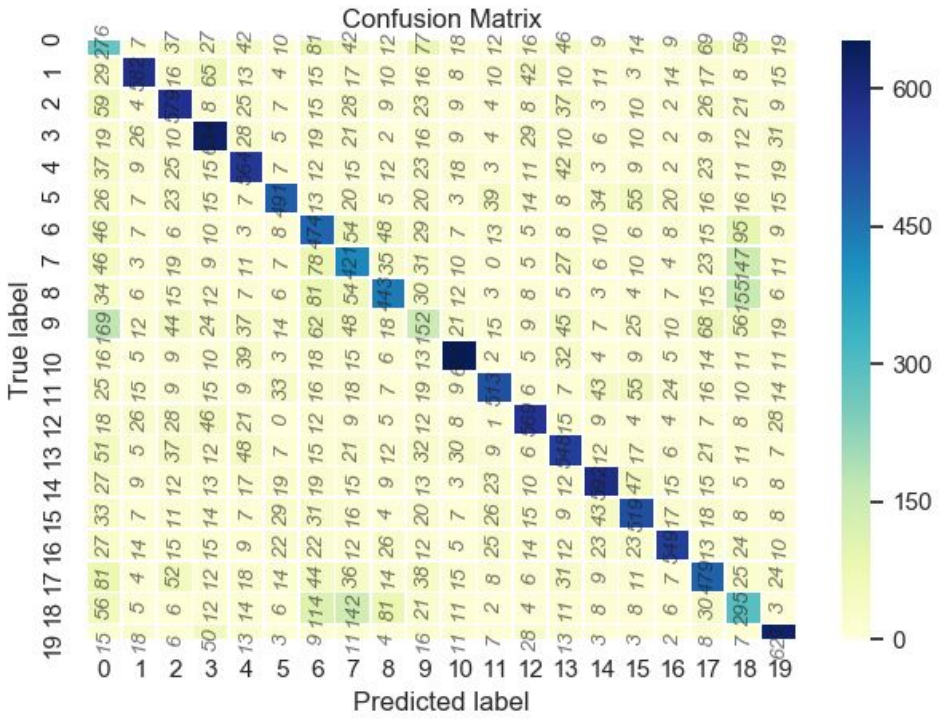
```
parameter_space = {
    "clf__fit_prior": [True, False],
    "clf__alpha": [0, 0.5, 0.25, 0.15, 0.05, 0.005, 1],
}
```

Parameter space for MultinomialNB

we noticed the value of hyperparameter alpha=0.25 results in the highest validation score (details appendix 1).

Confusion Matrix: Below illustrates confusion matrix between actual classes and predicted classes.

0= ask reddit
1= global offensive
2= music
3= overwatch
4= anime
5= baseball
6= canada
7= conspiracy
8= europe
9= funny
10= game of thrones
11= hockey
12= league of legends
13= movies
14= nba
15= nfl
16= soccer
17= trees
18= world news
19= wow



Confusion Matrix prediction/actual classes Naive bayes

As expected, high values are on the main diagonal, indicating most of the times the predicted class is the same as the actual class. Highest correct prediction, in order belongs to class 10 (“Game of Thrones”), class 3 (“Overwatch”), and class 19 (“wow”), with 652, 634 and 627 correct classifications respectively. The most misclassification is related to Class 9, “funny”, where number of correct

classification is only 152 with the most misclassification for group 0, “Ask Reddit”, with 169 misclassification. It is interesting that the second misclassified class is class 0, “Ask Reddit”, with 276 correct prediction and with the most misclassification of 169 for class 9, “funny”. After these two, class 18, “World News” have the highest misclassification with only 295 correct classifications.

Logistic Regression

Setting hyper parameters: Using exhaustive search of GridSearchCV we tried all different combinations of the following parameters:

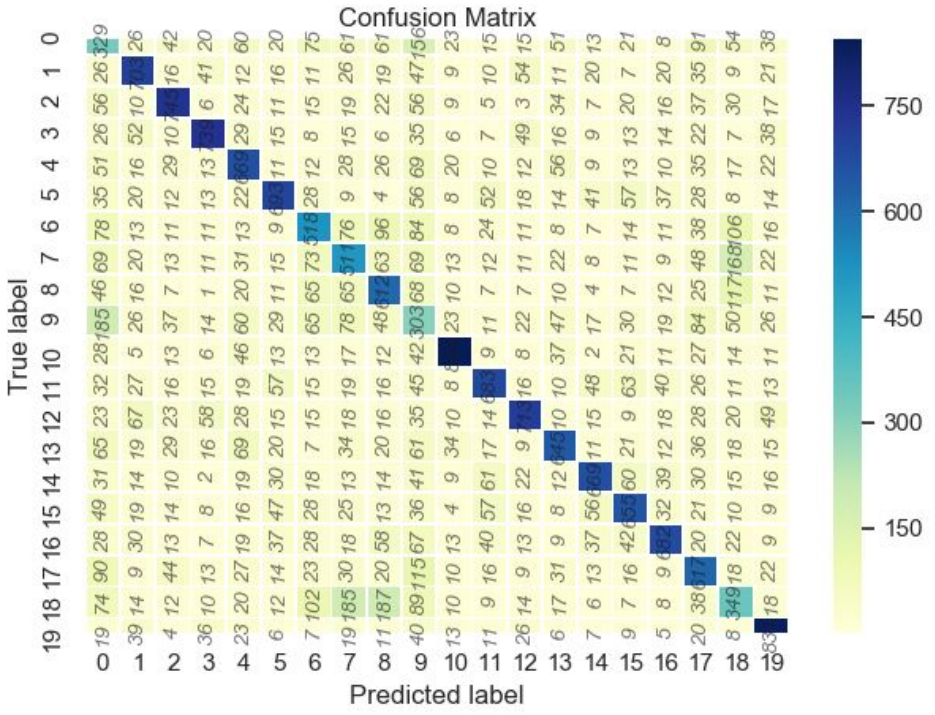
```
parameter_space = {
    "clf__penalty": ["l1", "l2"],
    "clf__fit_intercept": [True, False],
    "clf__multi_class": ['ovr', "auto"],
    "clf__random_state": [None, 5]
}
```

Parameter space for Logistic Regression

noticed the default values of hyperparameters results in the highest validation score (details appendix 2).

Confusion Matrix: Below illustrates confusion matrix between actual classes and predicted classes.

0= ask reddit
1= global offensive
2= music
3= overwatch
4= anime
5= baseball
6= canada
7= conspiracy
8= europe
9= funny
10= game of thrones
11= hockey
12= league of legends
13= movies
14= nba
15= nfl
16= soccer
17= trees
18= world news
19= wow



Confusion Matrix prediction/actual classes Logistic Regression

As expected, high values are on the main diagonal, indicating most of the times the predicted class is the same as the actual class. The most misclassification is related to Class 9, “funny”, where number of correct classification is 303 with the most misclassification for group 0, “Ask Reddit”, with 185 misclassification. After that, “World News” class has the highest misclassification, where 349 match found between the actual class and the predicted class and the most misclassification is related to classes 7, conspiracy, and 8, europe, with 185 and 187 misclassification respectively.

Multilayer Perceptron

Setting hyper parameters: Using exhaustive search of GridSearchCV we tried all different combinations of the following parameters:

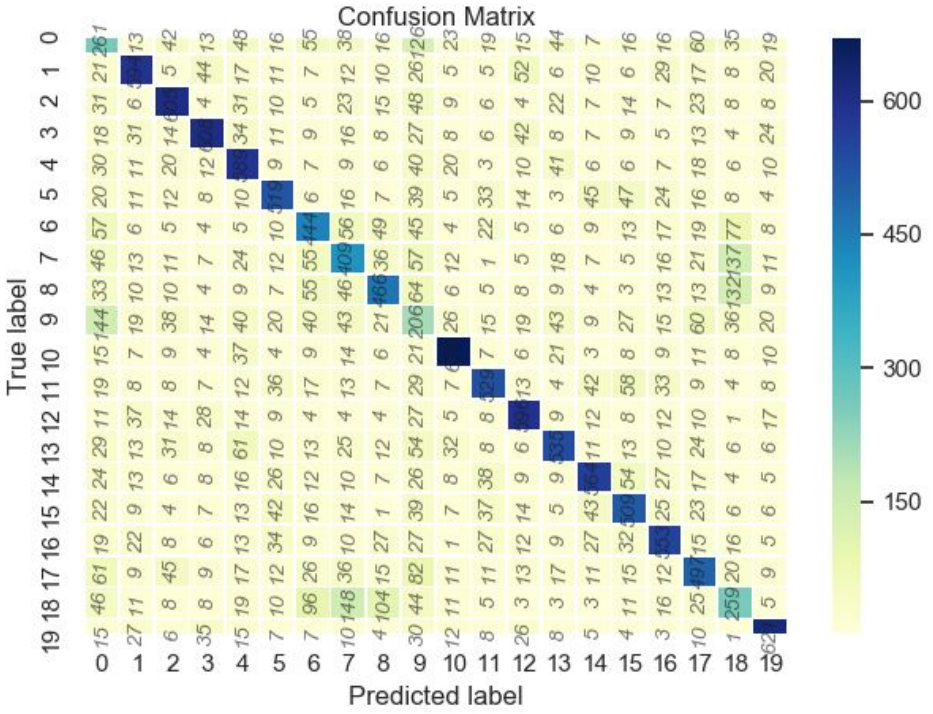
```
parameter_space = {
    "clf__hidden_layer_sizes": [(50, 50, 50), (50, 100, 50), (100,)],
    "clf__activation": ['tanh', 'relu'],
    "clf__solver": ['sgd', 'adam'],
    "clf__alpha": [0.0001, 0.05],
    "clf__learning_rate": ['constant', 'adaptive'],
}
```

Parameter space for Multilayer Perceptron

noticed the default values of hyperparameters results in the highest validation score (details appendix 3).

Confusion Matrix: Below illustrates confusion matrix between actual classes and predicted classes.

0= ask reddit
1= global offensive
2= music
3= overwatch
4= anime
5= baseball
6= canada
7= conspiracy
8= europe
9= funny
10= game of thrones
11= hockey
12= league of legends
13= movies
14= nba
15= nfl
16= soccer
17= trees
18= world news
19= wow



Confusion Matrix prediction/actual classes Multilayer Perceptron

As expected, high values are on the main diagonal, indicating most of the times the predicted class is the same as the actual class. Highest correct prediction, in order belongs to class 10 (“Game of Thrones”), class 4 (“anime”), and class 19 (“wow”), with 670, 621 and 608 correct classifications respectively. The most misclassification is related to Class 9, “funny”, where number of correct classification is only 206 with the most misclassification for group 0, “Ask Reddit”, with 144 misclassification. It is interesting that the second misclassified class is class 0, “Ask Reddit”, with 261 correct prediction and with the most misclassification of 144 for class 9, “funny”. After these two, class 18, “World News” have the highest misclassification with only 259 correct classifications.

Discussion

Models:

We used Naive Bayes model which is simple to implement and works well with high dimensions and computationally is fast. On the other hand it naively relies on the independence assumption and does not perform really well if the assumption is not true.

We also used this Logistic regression which is widely used across the industry and conveniently provides probabilities for the outcomes. On the other hand, it does not really perform well when the feature space is too large and it relies on transformation for non-linear features.

Hyper parameters optimization:

In this competition we have used exhaustive search for tuning our hyper parameters [6] which is computationally costly. We could improve this by changing to different methods such as random search, Bayesian or gradient based optimizations. Additionally we have

Statement of Contributions

Ryan Mokarian:

Ryan Contributed to the all milestones of the project. In the first milestone, he participated in Naive Bayes classifier modeling including coding for the Laplace smoothing. For the second and third milestones, he participated in modeling, data analysis and post processing of the results including preparation of the Confusion Matrices and Heatmaps and participation in writing the final report.

Maysam Mokarian:

Maysam contributed to all milestones of the eproject. Including Naive bayes with and without Laplace smoothing. Maysam Also contributed in the feature engineering and data preprocessing analysis. Model and data analysis and generating reports and results. Maysam Also contributed in generating the box plots in this report as well as running and generating exhaustive search for hyperparameter optimization.

We (**Ryan Mokarian** and **Maysam Mokarian**), hereby state that all the work presented in this report is that of the authors.

References

[1] “Stop Words.” *Wikipedia*, Wikimedia Foundation, 25 Oct. 2019, en.wikipedia.org/wiki/Stop_words.

[2] “Nltk.tokenize Package.” *Nltk.tokenize Package - NLTK 3.4.5 Documentation*,

www.nltk.org/api/nltk.tokenize.html.

[3] *Accessing Text Corpora and Lexical Resources*, www.nltk.org/book/ch02.html.

[4] “Tf–Idf.” *Wikipedia*, Wikimedia Foundation, 5 Nov. 2019, en.wikipedia.org/wiki/Tf–idf.

[5] “5.2. Feature Extraction.” *Scikit*, scikit-learn.org/stable/modules/feature_extraction.html.

[6] “3.2. Tuning the Hyper-Parameters of an Estimator.” *Scikit*, scikit-learn.org/stable/modules/grid_search.html.

[7] “Multilayer Perceptron.” *Wikipedia*, Wikimedia Foundation, 26 Oct. 2019, en.wikipedia.org/wiki/Multilayer_perceptron#cite_ref-1.

Appendix 1

MultiLayer perceptron exhaustive search results:

Result of exhaustive search of GridSearchCV for Naive Bayes.
Best parameters found is as following:
{'clf__alpha': 0.25, 'clf__fit_prior': False}

	accuracy	alpha	fit_prior
0	0.467 (+/-0.003)	clf__alpha: 0	clf__fit_prior: True
1	0.467 (+/-0.002)	clf__alpha: 0	clf__fit_prior: False
2	0.545 (+/-0.006)	clf__alpha: 0.5	clf__fit_prior: True
3	0.546 (+/-0.006)	clf__alpha: 0.5	clf__fit_prior: False
4	0.546 (+/-0.003)	clf__alpha: 0.25	clf__fit_prior: True
5	0.547 (+/-0.003)	clf__alpha: 0.25	clf__fit_prior: False
6	0.545 (+/-0.003)	clf__alpha: 0.15	clf__fit_prior: True
7	0.545 (+/-0.002)	clf__alpha: 0.15	clf__fit_prior: False
8	0.536 (+/-0.002)	clf__alpha: 0.05	clf__fit_prior: True
9	0.536 (+/-0.003)	clf__alpha: 0.05	clf__fit_prior: False
10	0.515 (+/-0.001)	clf__alpha: 0.005	clf__fit_prior: True
11	0.514 (+/-0.001)	clf__alpha: 0.005	clf__fit_prior: False
12	0.539 (+/-0.005)	clf__alpha: 1	clf__fit_prior: True
13	0.539 (+/-0.005)	clf__alpha: 1	clf__fit_prior: False

Appendix 2

Logistic Regression exhaustive search:

Result of exhaustive search of GridSearchCV for Logistic Regression
Best parameters found is as following:
{'clf__fit_intercept': False, 'clf__multi_class': 'ovr', 'clf__penalty': 'l2', 'clf__random_state': None}

	accuracy	fit_intercept	multi_class	penalty	random state
0	0.492 (+/-0.003)	clf__fit_intercept: True	clf__multi_class: ovr	clf__penalty: l1	clf__random_state: None
1	0.492 (+/-0.003)	clf__fit_intercept: True	clf__multi_class: ovr	clf__penalty: l1	clf__random_state: 5
2	0.523 (+/-0.005)	clf__fit_intercept: True	clf__multi_class: ovr	clf__penalty: l2	clf__random_state: None
3	0.523 (+/-0.005)	clf__fit_intercept: True	clf__multi_class: ovr	clf__penalty: l2	clf__random_state: 5
4	0.492 (+/-0.003)	clf__fit_intercept: True	clf__multi_class: auto	clf__penalty: l1	clf__random_state: None
5	0.492 (+/-0.003)	clf__fit_intercept: True	clf__multi_class: auto	clf__penalty: l1	clf__random_state: 5
6	0.523 (+/-0.005)	clf__fit_intercept: True	clf__multi_class: auto	clf__penalty: l2	clf__random_state: None
7	0.523 (+/-0.005)	clf__fit_intercept: True	clf__multi_class: auto	clf__penalty: l2	clf__random_state: 5
8	0.467 (+/-0.005)	clf__fit_intercept: False	clf__multi_class: ovr	clf__penalty: l1	clf__random_state: None
9	0.467 (+/-0.005)	clf__fit_intercept: False	clf__multi_class: ovr	clf__penalty: l1	clf__random_state: 5
10	0.527 (+/-0.007)	clf__fit_intercept: False	clf__multi_class: ovr	clf__penalty: l2	clf__random_state: None
11	0.527 (+/-0.007)	clf__fit_intercept: False	clf__multi_class: ovr	clf__penalty: l2	clf__random_state: 5
12	0.467 (+/-0.005)	clf__fit_intercept: False	clf__multi_class: auto	clf__penalty: l1	clf__random_state: None
13	0.467 (+/-0.005)	clf__fit_intercept: False	clf__multi_class: auto	clf__penalty: l1	clf__random_state: 5
14	0.527 (+/-0.007)	clf__fit_intercept: False	clf__multi_class: auto	clf__penalty: l2	clf__random_state: None
15	0.527 (+/-0.007)	clf__fit_intercept: False	clf__multi_class: auto	clf__penalty: l2	clf__random_state: 5

Appendix 3

MultiLayer perceptron exhaustive search:

Result of exhaustive search of GridSearchCV for Multilayer perceptron.
Best parameters found is as following:
{'clf__activation': 'relu', 'clf__alpha': 0.0001, 'clf__hidden_layer_sizes': (100,), 'clf__learning_rate': 'constant', 'clf__solver': 'adam'}

	accuracy	activation	alpha	hidden layer	learning rate	solver
0	0.252 (+/-0.056)	clf__activation: tanh	clf__alpha: 0.0001	clf__hidden_layer_sizes: (50, 50, 50)	clf__learning_rate: constant	clf__solver: sgd
1	0.498 (+/-0.005)	clf__activation: tanh	clf__alpha: 0.0001	clf__hidden_layer_sizes: (50, 50, 50)	clf__learning_rate: constant	clf__solver: adam
2	0.246 (+/-0.009)	clf__activation: tanh	clf__alpha: 0.0001	clf__hidden_layer_sizes: (50, 50, 50)	clf__learning_rate: adaptive	clf__solver: sgd
3	0.505 (+/-0.004)	clf__activation: tanh	clf__alpha: 0.0001	clf__hidden_layer_sizes: (50, 50, 50)	clf__learning_rate: adaptive	clf__solver: adam
4	0.254 (+/-0.039)	clf__activation: tanh	clf__alpha: 0.0001	clf__hidden_layer_sizes: (50, 100, 50)	clf__learning_rate: constant	clf__solver: sgd
5	0.510 (+/-0.014)	clf__activation: tanh	clf__alpha: 0.0001	clf__hidden_layer_sizes: (50, 100, 50)	clf__learning_rate: constant	clf__solver: adam
6	0.265 (+/-0.011)	clf__activation: tanh	clf__alpha: 0.0001	clf__hidden_layer_sizes: (50, 100, 50)	clf__learning_rate: adaptive	clf__solver: sgd
7	0.508 (+/-0.011)	clf__activation: tanh	clf__alpha: 0.0001	clf__hidden_layer_sizes: (50, 100, 50)	clf__learning_rate: adaptive	clf__solver: adam
8	0.354 (+/-0.005)	clf__activation: tanh	clf__alpha: 0.0001	clf__hidden_layer_sizes: (100,)	clf__learning_rate: constant	clf__solver: sgd
9	0.556 (+/-0.003)	clf__activation: tanh	clf__alpha: 0.0001	clf__hidden_layer_sizes: (100,)	clf__learning_rate: constant	clf__solver: adam
10	0.352 (+/-0.020)	clf__activation: tanh	clf__alpha: 0.0001	clf__hidden_layer_sizes: (100,)	clf__learning_rate: adaptive	clf__solver: sgd
11	0.557 (+/-0.001)	clf__activation: tanh	clf__alpha: 0.0001	clf__hidden_layer_sizes: (100,)	clf__learning_rate: adaptive	clf__solver: adam
12	0.269 (+/-0.020)	clf__activation: tanh	clf__alpha: 0.05	clf__hidden_layer_sizes: (50, 50, 50)	clf__learning_rate: constant	clf__solver: sgd
13	0.491 (+/-0.003)	clf__activation: tanh	clf__alpha: 0.05	clf__hidden_layer_sizes: (50, 50, 50)	clf__learning_rate: constant	clf__solver: adam
14	0.241 (+/-0.005)	clf__activation: tanh	clf__alpha: 0.05	clf__hidden_layer_sizes: (50, 50, 50)	clf__learning_rate: adaptive	clf__solver: sgd
15	0.491 (+/-0.008)	clf__activation: tanh	clf__alpha: 0.05	clf__hidden_layer_sizes: (50, 50, 50)	clf__learning_rate: adaptive	clf__solver: adam
16	0.251 (+/-0.054)	clf__activation: tanh	clf__alpha: 0.05	clf__hidden_layer_sizes: (50, 100, 50)	clf__learning_rate: constant	clf__solver: sgd

17	0.499 (+/-0.007)	clf__activation: tanh	clf__alpha: 0.05	clf__hidden_layer_sizes: (50, 100, 50)	clf__learning_rate: constant	clf__solver: adam
18	0.261 (+/-0.035)	clf__activation: tanh	clf__alpha: 0.05	clf__hidden_layer_sizes: (50, 100, 50)	clf__learning_rate: adaptive	clf__solver: sgd
19	0.499 (+/-0.016)	clf__activation: tanh	clf__alpha: 0.05	clf__hidden_layer_sizes: (50, 100, 50)	clf__learning_rate: adaptive	clf__solver: adam
20	0.344 (+/-0.023)	clf__activation: tanh	clf__alpha: 0.05	clf__hidden_layer_sizes: (100,)	clf__learning_rate: constant	clf__solver: sgd
21	0.537 (+/-0.009)	clf__activation: tanh	clf__alpha: 0.05	clf__hidden_layer_sizes: (100,)	clf__learning_rate: constant	clf__solver: adam
22	0.334 (+/-0.029)	clf__activation: tanh	clf__alpha: 0.05	clf__hidden_layer_sizes: (100,)	clf__learning_rate: adaptive	clf__solver: sgd
23	0.539 (+/-0.007)	clf__activation: tanh	clf__alpha: 0.05	clf__hidden_layer_sizes: (100,)	clf__learning_rate: adaptive	clf__solver: adam
24	0.146 (+/-0.145)	clf__activation: relu	clf__alpha: 0.0001	clf__hidden_layer_sizes: (50, 50, 50)	clf__learning_rate: constant	clf__solver: sgd
25	0.469 (+/-0.006)	clf__activation: relu	clf__alpha: 0.0001	clf__hidden_layer_sizes: (50, 50, 50)	clf__learning_rate: constant	clf__solver: adam
26	0.090 (+/-0.102)	clf__activation: relu	clf__alpha: 0.0001	clf__hidden_layer_sizes: (50, 50, 50)	clf__learning_rate: adaptive	clf__solver: sgd
27	0.473 (+/-0.013)	clf__activation: relu	clf__alpha: 0.0001	clf__hidden_layer_sizes: (50, 50, 50)	clf__learning_rate: adaptive	clf__solver: adam
28	0.050 (+/-0.002)	clf__activation: relu	clf__alpha: 0.0001	clf__hidden_layer_sizes: (50, 100, 50)	clf__learning_rate: constant	clf__solver: sgd
29	0.471 (+/-0.017)	clf__activation: relu	clf__alpha: 0.0001	clf__hidden_layer_sizes: (50, 100, 50)	clf__learning_rate: constant	clf__solver: adam
30	0.054 (+/-0.012)	clf__activation: relu	clf__alpha: 0.0001	clf__hidden_layer_sizes: (50, 100, 50)	clf__learning_rate: adaptive	clf__solver: sgd
31	0.468 (+/-0.010)	clf__activation: relu	clf__alpha: 0.0001	clf__hidden_layer_sizes: (50, 100, 50)	clf__learning_rate: adaptive	clf__solver: adam
32	0.217 (+/-0.228)	clf__activation: relu	clf__alpha: 0.0001	clf__hidden_layer_sizes: (100,)	clf__learning_rate: constant	clf__solver: sgd
33	0.558 (+/-0.002)	clf__activation: relu	clf__alpha: 0.0001	clf__hidden_layer_sizes: (100,)	clf__learning_rate: constant	clf__solver: adam
34	0.335 (+/-0.014)	clf__activation: relu	clf__alpha: 0.0001	clf__hidden_layer_sizes: (100,)	clf__learning_rate: adaptive	clf__solver: sgd
35	0.556 (+/-0.001)	clf__activation: relu	clf__alpha: 0.0001	clf__hidden_layer_sizes: (100,)	clf__learning_rate: adaptive	clf__solver: adam
36	0.052 (+/-0.003)	clf__activation: relu	clf__alpha: 0.05	clf__hidden_layer_sizes: (50, 50, 50)	clf__learning_rate: constant	clf__solver: sgd
37	0.447 (+/-0.010)	clf__activation: relu	clf__alpha: 0.05	clf__hidden_layer_sizes: (50, 50, 50)	clf__learning_rate: constant	clf__solver: adam
38	0.097 (+/-0.107)	clf__activation: relu	clf__alpha: 0.05	clf__hidden_layer_sizes: (50, 50, 50)	clf__learning_rate: adaptive	clf__solver: sgd
39	0.442 (+/-0.011)	clf__activation: relu	clf__alpha: 0.05	clf__hidden_layer_sizes: (50, 50, 50)	clf__learning_rate: adaptive	clf__solver: adam

40	0.054 (+/-0.012)	clf__activation: relu	clf__alpha: 0.05	clf__hidden_layer_sizes: (50, 100, 50)	clf__learning_rate: constant	clf__solver: sgd
41	0.451 (+/-0.004)	clf__activation: relu	clf__alpha: 0.05	clf__hidden_layer_sizes: (50, 100, 50)	clf__learning_rate: constant	clf__solver: adam
42	0.053 (+/-0.007)	clf__activation: relu	clf__alpha: 0.05	clf__hidden_layer_sizes: (50, 100, 50)	clf__learning_rate: adaptive	clf__solver: sgd
43	0.443 (+/-0.021)	clf__activation: relu	clf__alpha: 0.05	clf__hidden_layer_sizes: (50, 100, 50)	clf__learning_rate: adaptive	clf__solver: adam
44	0.220 (+/-0.240)	clf__activation: relu	clf__alpha: 0.05	clf__hidden_layer_sizes: (100,)	clf__learning_rate: constant	clf__solver: sgd
45	0.537 (+/-0.007)	clf__activation: relu	clf__alpha: 0.05	clf__hidden_layer_sizes: (100,)	clf__learning_rate: constant	clf__solver: adam
46	0.335 (+/-0.046)	clf__activation: relu	clf__alpha: 0.05	clf__hidden_layer_sizes: (100,)	clf__learning_rate: adaptive	clf__solver: sgd
47	0.534 (+/-0.001)	clf__activation: relu	clf__alpha: 0.05	clf__hidden_layer_sizes: (100,)	clf__learning_rate: adaptive	clf__solver: adam