Compararing analytic machine learning techniques and multilayer perceptron neural networks to predict Yelp Review star rating

Maxime Chabance (583912) maxime.chabance@aalto.fi

Hugues Verlin (584788) hugues.verlin@aalto.fi

Abstract

This paper explore several machine learning techniques in order to solve the two problems of the Yelp data challenge. A regression and a classification problem are studied in depth. For each of these, a comparison of several algorithms is performed. Feature selections is also studied and performed on the dataset. Experimentations with Multilayer Perceptron Neural Networks are done and allow us to get the best results, both in regression and classification. This paper also features our own implementation of two algorithms used in machine learning: a Linear Regressor and K-Nearest Neighbors classifier. The implementations are written in Python and use the machine learning library `scikit-learn`.

1. Introduction

Yelp is a website and application permitting users to write reviews about different places or services (restaurants, shops ...). Each review consists of a short text and a rate between 0 and 5. Then users can consult those reviews to choose the one place to go.

We will try to answer two problems.

1. Predict the number of 'useful votes' the review has received.

2. Predict if the review is positive (grade higher or equal to 3)

Yelp wants to distinguish itself from other similar websites by weighting user comments according to its usefulness. The pertinence of a comment is determined by the votes of other users. Yelp would like to predict if a comment is useful using machine learning. That way, it would be able to sort comments immediately, no need to wait the votes of tens of users.

As for the second question, it could be interesting to know if a review is positive without having the user to grade the place. It could result in more natural interactions with the application. The users just write what they think and Yelp will infer the grade.

2. Methods

2.1 Input data

The data we got is already preprocessed and given as bag of 50 manually selected words. The 50 words are:
{ 'but' 'good' 'place' 'food' 'great' 'very' 'service' 'back' 'really' 'nice'
'love' 'little' 'ordered' 'first' 'much' 'came' 'went' 'try' 'staff' 'people'
'restaurant' 'order' 'never' 'friendly' 'pretty' 'come' 'chicken' 'again' 'vegas'
'definitely' 'menu' 'better' 'delicious' 'experience' 'amazing' 'wait' 'fresh'
'bad' 'price' 'recommend' 'worth' 'enough' 'customer' 'quality' 'taste' 'atmosphere'
'however' 'probably' 'far' 'disappointed' }

The data is represented as a matrix of frequencies for each word in each message. That's a very simple way of representing this kind of data. We surely lost a lot of information this way. For instance, the world 'but' out of its context doesn't mean a lot. Moreover, the adjective 'good' could either be positive or negative if preceded by 'not'.

There are other representations of natural language texts a lot more complex. One of the most famous is word2vec [2], a group of neural network models which process large chunk of text into high dimensional vectors. Those vectors are supposed to capture the semantic of the text. This could have yield to a more difficult task, but could have given better results.

To solve the two problems, we have tried many algorithms. The general idea is to use a feature selection technique and then to perform the fitting using the best available algorithm. We have a cross-validation mechanism to help us seeking the best algorithm. We have also implemented our own linear regressor and a K-Nearest Neighbors classifier.

2.2 Pipeline

As a lot of pre-processing has already been made on the Yelp dataset, we only need to perform a feature selection before using the algorithms. The pipeline used for the project is the following:

1. Feature selection
2. Algorithm to perform a regression or a classification
3. Cross-validating the results

2.3 Feature Selection

The feature selection process is important because it can lead to a good improvement in performance when using the algorithm. Indeed, decreasing the number of features allows us to decrease the dimensionality of the problem, which permits us to perform faster training.

Also, a good selection of feature can prevent over-fitting, as it will remove the ones that do not bring enough information for the prediction.

We will then present the two methods that we used to perform the features selection.

2.3.1 Select K-Best

This method allows to select features according to their k highest score. This score is calculated using `f_regression` for the regression problem and `chi2` for the classification problem [1].

`f_regression` is a quick linear model for testing the effect of a single regressor, sequentially for many regressors. This is done in 2 steps: [4]

- The cross correlation between each regressor and the target is computed, that is

$$\frac{(\mathbf{X}[:, i] - \text{mean}(\mathbf{X}[:, i]))(\mathbf{y} - \text{mean}(\mathbf{y}))}{\text{std}(\mathbf{X}[:, i]) \times \text{std}(\mathbf{y})}$$

- It is converted to an *F-score* then to a *p-value*.

`chi2` is a test that computes *chi-squared* stats between each non-negative feature and class. The *chi-square* test measures dependence between stochastic variables, and therefore allows us to remove the features that are the most likely to be independent of the class and so irrelevant for the classification. [3]

2.3.2 Feature ranking with recursive feature elimination

The recursive feature elimination is another method of feature selection. The goal here is to recursively remove the less important feature. At each step, every subset of $n - 1$ features are tried, and the one with the best score is kept. And so on and forth until we obtain the wanted number of features.

2.4 Regression algorithms

2.4.1 Linear Regression

One the most simple algorithm used here is the linear regression. It is the first one that is we have tested because of it simplicity, and allows us to see how far from linearity the dataset can be. The idea of linear regression is to find a linear mapping between the features matrix and the target vector. We have used the Ordinary least squares technique to fit the data.

2.4.2 Multilayer Perceptron Network Regressor

Since the linear model was quite good as discussed in the last part of the paper, we have also tried several other techniques. The one that we have found to perform really well is presented here.

A multilayer perceptron (MLP) is a feedforward artificial neural network model that maps sets of input data onto a set of outputs. It is made of several layers that are connected to each other. Except for the input nodes, each node is a neuron (or processing element) with a nonlinear activation function. MLP uses a supervised learning technique called back-propagation for training the network. The advantages of using An MLP is that it will allow us to find a non-linear mapping between the data features and the target.

The implementation of the MLP used here is the one that comes with sckit-learn. This one cannot be used to compute on GPU but it was not a problem considering the size of the dataset.

The main problem that comes with MLP is the number of hyper-parameters. It requires a lot of tuning to perform correctly. The answer to this problem is to used cross-validation to test a great number of them.

2.5 Classification algorithms

2.5.1 K-Nearest Neighbors Classifier

The K-NN Classifier [5] is a very simple technique that looks for the k-nearest known features vectors and to choose the best represented class. The model is "fitted" just by storing known features vectors. Then the prediction is done by searching the nearest vectors among the known dataset.

2.5.2 Logistic Regression Classifier

Logistic regression classifier [7] is one of the oldest algorithm in the area of machine learning. It is close to linear regression, with some differences:

- It is a binary classification model
- It makes use of the Bernouilli distribution instead of the Gaussian distribution
- It clamps the result value between 0 and 1 using the logistic function (See figure 1)

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

.

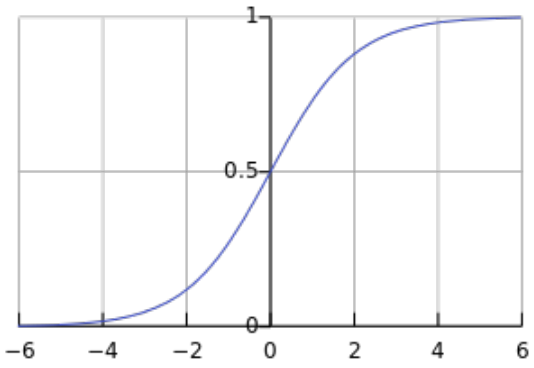

Figure 1: The standard logistic function $\sigma(t)$

The result represents the probability for a point to be of class 1. The class is then chosen by rounding this probability.

2.5.3 Multilayer Perceptron Network Classifier

We also applied the MLP to classification. Its works in a similar fashion as the one used for regression. The main difference is that is it just maps the result to one of the class according to the different probabilities it has computed.

3. Experiments

3.1 Cross-validation

In order to select the best method for the project, we have used extensively the `GridSearchCV` class of `sk_learn`. This allows us to test any number of methods using cross-validation. The best technique is then trained on the whole dataset. After trial and errors of the methods described before, we have tried many hyperparameters for the `MLP Regressor`. The code is available in the appendice.

The score for the *regression* defined as followed [1]

$$\text{Score} = 1 - \frac{\sum_i (y_{t,i} - y_{p,i})^2}{\sum_i (y_{t,i} - y_M)^2}$$

where y_M is the mean of the y target.

A constant model that always predicts the expected value of y, disregarding the input features, would get a score of 0.0.

The score for the *classification* is simply the percentage of correctly classified inputs.

We have performed the cross-validation with a `K-fold` algorithm (5-folds in our case). We could have used `LOOCV` for better accuracy but it would have take too long to compute on our computer.

3.2 Regression

3.2.1 Linear Regressor

We have made a simple linear regressor using ordinary least square. The following formula has been used for the linear regressor: [6]

$$\mathbf{C} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} = \left(\sum \mathbf{x}_i \mathbf{x}_i^{\mathrm{T}}\right)^{-1} \left(\sum \mathbf{x}_i y_i\right).$$

where $\mathbf{X}$ is the features matrix and $\mathbf{y}$ is the target vector.

The prediction vector $\mathbf{P}$ is then computed using:

$$\mathbf{P} = \mathbf{C}\mathbf{y}$$

The python code implementing the linear regressor is attached in the appendix A.

3.2.2 Multilayer Perceptron Network Regressor

The multilayer perceptron network used during the experiment is the one implemented in the `sckit-learn` library. It can take a large number of different parameters

We ended up using a MLP with a single hidden layer of 100 neurons, after a lot of trial and errors. We have tried several version of the MLP using either the tanh function or the `rectified linear unit function`. Several values for the L2 penalty and the learning rate have also been tested.

The MLP need also the features to be scaled before being used. We have standardized the features by removing the mean and scaling to unit variance using the scaler of `sckit-learn`.

3.3 Classification

3.3.1 K-Nearest Neighbors

We implemented our own algorithm to compute the KNN-classifier. You can find the code in appendix B.

This is a naive implementation, a sort is done for each feature vector. This makes the computation a bit slow but it achieves its theoretical value.

3.3.2 Logistic Regression Classifier

We make use of `LogisticRegression` With `Pipeline` and `GridSearchCV`, we could test several values for the different parameters of the estimator and the feature selection.

3.3.3 Multilayer Perceptron Network Classifier

Same as for the regression, we used the classifier `MLPClassifier` inside of `GridSearchCV` to try a lot of parameter configurations.

4. Results

4.1 Regression

4.1.1 Feature selection

The results of the feature selection for `SelectKBest` is presented on the figure 2. We can see that the top seven features are more important than the others. However, we have found during the cross-validation that it was more accurate to use ten features.

We have also perform a cross-validation using the `RFE` (Recursive Features Elimination). This have yield the following set of seven features: `['food', 'service', 'staff', 'restaurant', 'menu', 'price', 'taste']` . One can notice that these are the same features that are returned by `Select-K-Best`.

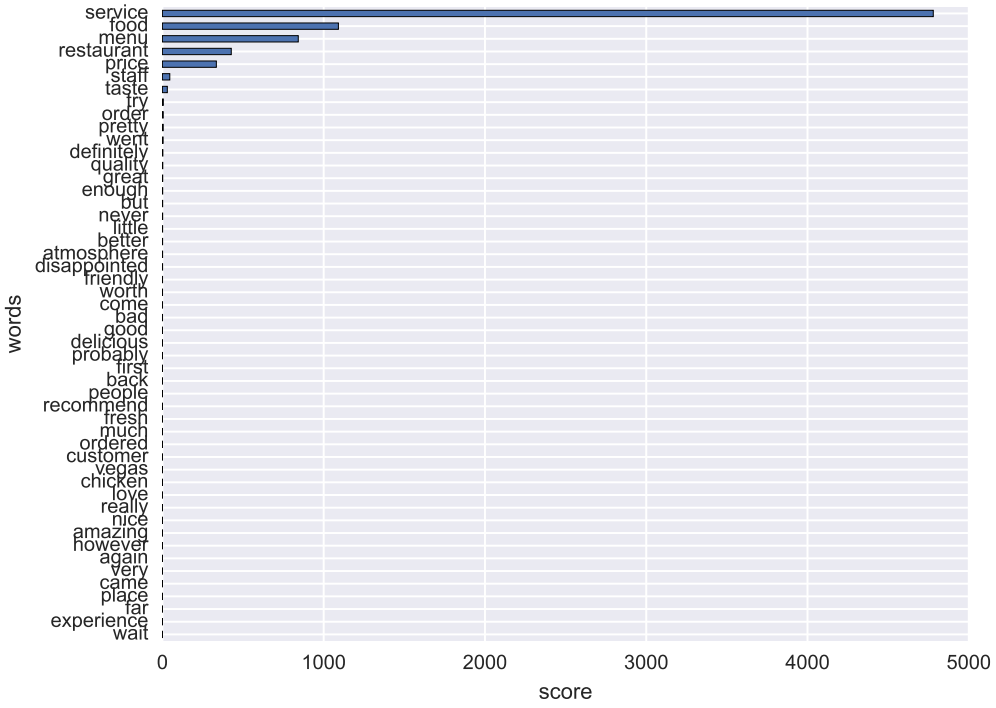

Figure 2: Result of the feature selection with Select K-Best

4.1.2 Results summary

Linear regression The best results for the linear regressor are shown in the figure 3. The features used are the seven ones that were selected by the `RFE` algorithm.

The mean squared error (MSE) on the test dataset is: `0.047664`.

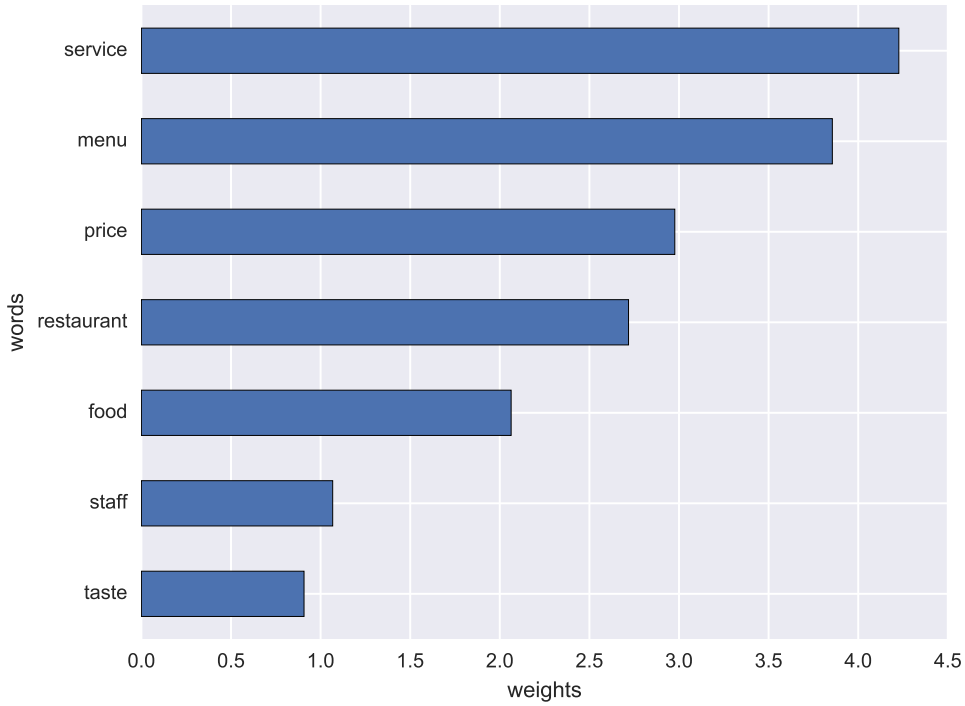

Figure 3: Weights of the coefficient after fitting the linear regressor

Mlp Regressor Using our best model, the `MLP Regressor`, we have obtain a score of 0.999413 during the training phase. The following parameters are used:

- The 10 most useful features are kept
- 1 hidden layer of 100 neurons
- activation function: rectified linear unit function [8], returns $f(x) = max(0, x)$
- solver: `sgd` (Stochastic Gradient Descent [9])
- learning rate: `0.1`
- L2 penalty (regularization term) parameter: `0.1`

Test results:

- The mean squared error (MSE) on the test dataset is : `0.005347`.
- The score obtained on `Kaggle` (after the submission deadline) is: `0.00626`.

4.2 Classification

4.2.1 Feature selection

We can see on the figure 5 that some words have a lot more importance than others. We select the first 14 words:

{'never' 'disappointed' 'but' 'delicious' 'pretty' 'definitely' 'great' 'bad' 'amazing' 'friendly' 'good' 'love' 'fresh' 'nice'}

4.2.2 Results summary

Here are the best validation accuracy obtained by the different methods:

K-Nearest Neighbors. Using our self-implemented model, we obtained an accuracy of 65 %

Logistic Regression Classifier. The best accuracy is 71.4 % obtained with the selected 14 features above. The weights for each word are visible on figure 4.

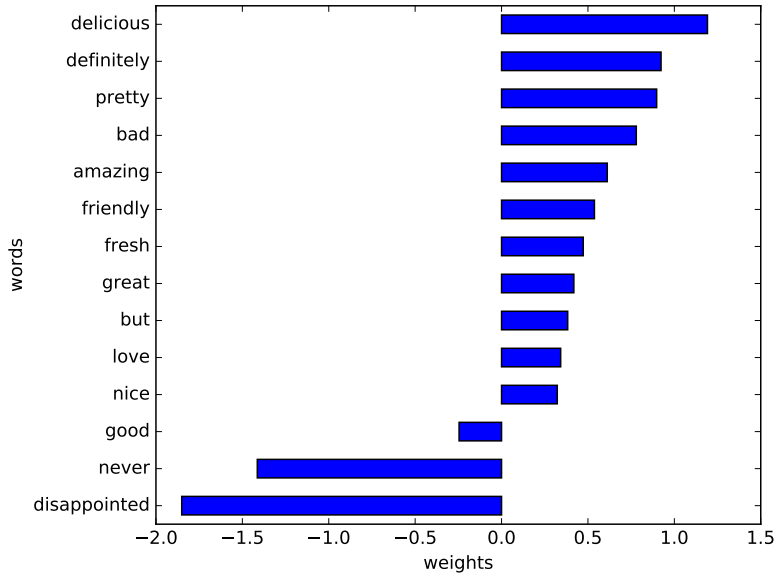

Figure 4: Impact of the different words on the probability for the review to be positive using the Logistic Regression

Multi Layer Perceptron Classifier. The best accuracy obtained is 0.71.5 %.
The following parameters are used:

- 1 hidden layer of 100 neurons
- activation function: *tanh*

- solver: `sgd` (Stochastic Gradient Descent [9])
- adaptive learning rate, initial value: 0.1
- L2 penalty (regularization term) parameter: `0.1`

Summary: The best models seem to be the Logistic Regression Classifier and the Multilayer Perceptron Classifier with provide almost the same accuracy.

You can find the code of the best classifier in appendix D.

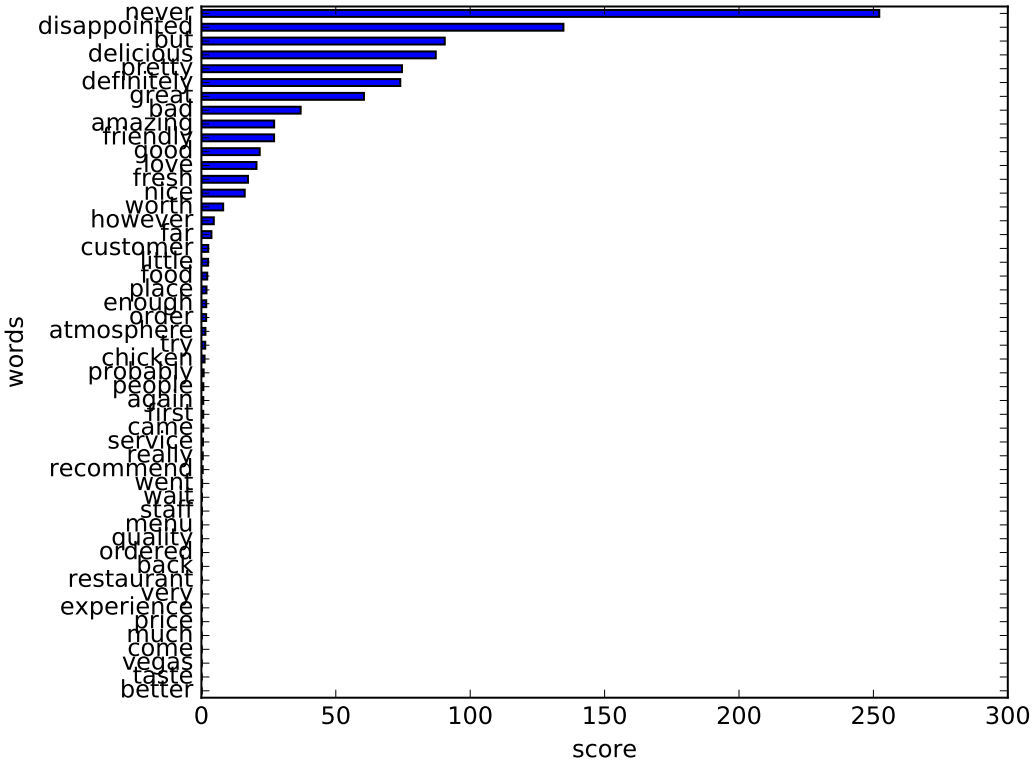

Figure 5: Result of the feature selection with Select K-Best for the classification

5. Discussion/Conclusions

The accuracy on the test data for the classification was 71%. This is not very good (only 21% more than the random classifier). This score is too low to allow us to trust the classifier. This low score can be explained by the simplistic data representation and the impossibility to distinguish negative from positive sentences.

This last point shows in some weights (figure 4) which seem counter-intuitive: bad is positive whereas good is negative.

The KNN-Classifier has been penalized by the sparsity of the feature vectors. Indeed, the distance is actually very similar for two very very different vectors. Sometimes, a quarter of the vectors are at equal distance from the reference vector. This means KNN-Classifier isn't appropriate for this kind of problems.

We have found that the results for the regression problem to be surprisingly good. Indeed, we tend to think that the number of votes on a comment do not depend only on the content but also on which page it is linked to. A page that is more visited that an other one will tend to have more votes on its comments. That's why the accuracy obtained seems almost too good. It can also be notice that this exercise would be very difficult for a human.

Not every techniques have been tested and could be considered in future studies. For instance, random forest boosting [10] is often very efficient on `Kaggle` challenges, for all king of problems.

References

[1] sk learn. Sk-learn, november 2016. http://scikit-learn.org.

[2] Wikipedia. Word2vec. https://en.wikipedia.org/wiki/Word2vec.

[3] Wikipedia. Chi-squared test, 2016. https://en.wikipedia.org/wiki/Chi-squared_test.

[4] Wikipedia. F-test, 2016. https://en.wikipedia.org/wiki/F-test.

[5] Wikipedia. Knn-classifier, 2016. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[6] Wikipedia. Linear regression, 2016. https://en.wikipedia.org/wiki/Linear_regression.

[7] Wikipedia. Logistic regression, 2016. https://en.wikipedia.org/wiki/Logistic_regression.

[8] Wikipedia. Rectifier linear unit, 2016. https://en.wikipedia.org/wiki/Rectifier_(neural_networks).

[9] Wikipedia. Stochastic gradient descent, 2016. https://en.wikipedia.org/wiki/Stochastic_gradient_descent.

[10] Masamitsu Tsuchiya Yohei Mishina and Hironobu Fujiyoshi. Boosted random forest.

Appendix A. Linear Regressor - code

```
import numpy as np
from numpy.linalg import inv


class LinearRegressor:
    """
    Simple multivariate linear regression
    using ordinary least square
    """
    coefs = None

    def fit(self, X: np.matrix, y: np.matrix):
        tp_X = np.transpose(X)
        self.coefs = inv(tp_X * X) * tp_X * np.transpose(y)

    def predict(self, target: np.matrix):
        return target * self.coefs
```

Appendix B. KNN classifier - code

```
import numpy as np
import pandas as pd
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

def dist(a,b) -> float:
    """
    Euclidian distance between two vectors
    """
    return np.linalg.norm(a-b)


class KNNClassifier:
    def __init__(self, k:int=5):
        self.k = k
        self.model_features = None
        self.model_target = None

    def fit(self, X: np.matrix, y: np.matrix):
        self.model_features = X
        self.model_target = list(y)

    def predict(self, features: np.matrix) -> np.matrix:
        res = []

        for i in range(len(features)):
            dists = sorted([(dist(features[i],
              self.model_features[j]), j) for j in
                range(len(self.model_features))])
            classes = {}
            for (d, j) in dists[:self.k]:
                clazz = self.model_target[j]
                classes[clazz] = 1 +
                  (classes[clazz] if clazz in classes else 0)

            res.append([max(classes.items(),
              key=lambda p: p[1])[0]])

        return np.matrix(res)
```

Appendix C. Code that produces the best Regression score

```
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
import pandas as pd

def constraints(val):
    return 0 if val < 0 else val


def generate_submission_regression(features, target, testing_data):
    scaler = StandardScaler()
    scaler.fit(features)
    features = scaler.transform(features)

    dim = SelectKBest(f_regression, k=10)
    features = dim.fit_transform(features, target)

    mlp = MLPRegressor(
        activation='relu',
        alpha=0.10000000000000001,
        batch_size='auto',
        beta_1=0.9,
        beta_2=0.999,
        epsilon=1e-08,
        hidden_layer_sizes=(100,),
        learning_rate='adaptive',
        learning_rate_init=0.10000000000000001,
        solver='sgd',
        tol=0.0001,
    )

    mlp.fit(features, target)

    testing_data = scaler.transform(testing_data)
    testing_data = dim.transform(testing_data)

    predictions = mlp.predict(testing_data)
    predictions = list(map(constraints, predictions))
    pd.DataFrame(predictions).to_csv('kaggle/results.csv')
```

Appendix D. Code that produces the best Classification score

```
import pandas as pd
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.neural_network import MLPClassifier


def best_classification(
  training_features, training_target, question_features):

    dim_reduction = SelectKBest(score_func=chi2, k=14)\
      .fit(training_features, training_target)
    training_features = dim_reduction.transform(training_features)
    question_features = dim_reduction.transform(question_features)

    classifier = MLPClassifier(alpha=0.1,
        learning_rate='adaptive',
        learning_rate_init='0.01',
        solver='sgd',
        activation='tanh')

    classifier.fit(training_features, training_target)

    return classifier.predict(question_features)
```