Machine Learning Report

Jamie Hall(jah79)

Introduction

This report will look into using two machine learning methods to be trained and used to predict labels for data from records of an input dataset. Kaggle was used as the main IDE for this project. This report will explain the creation of the algorithms, design choices, data pre-processing and any experimenting/testing executed.

Data Observations and Pre-processing

Upon observing the data in both datasets, there appeared to be 4 different labels for the data records. ‘N’, ‘O’, ‘A’ and ‘~’. This means the labels are in text form and will likely need to be converted into numerical data to work with the machine learning methods. Research was conducted online with various sources and from these I decided to use a label encoder to complete this task. A label encoder gives each unique label a numerical value. In this case, after conversion:

A = 0
N = 1
O = 2
~ = 3

The code used to convert the labels is as follows:

```
labelencoder = LabelEncoder()
y = labelencoder.fit_transform(y)
```

An extra function had to be used with labels for the deep-learning method. This np_utils function created dummy labels from the already numbered labels which is then used in the main machine learning code. The line to use this function is below:

```
dummy_y = np_utils.to_categorical(y)
```

Some values in the dataset were missing or invalid. The solution used for this problem replaces any empty slots with a value of ‘0’, although the mean value of that fields values could be used instead. I decided to use ‘0’ as faking a significant value based on the other fields could negatively impact the results whereas the field still has the same value as when it was empty by using a zero, it just has a numerical form now. As the data from the dataset is read in to a 2d array, the fillna() function is used to replace any ‘None’ or ‘na’ data.

For the two training sets to be used to train the machine learning methods, there is an ‘ID’ field which will have to be removed before any calculations and the ‘Type’ field which contains the labels mentioned above. The rest of the fields are all feature field, all containing numerical data as floats.

For the two test datasets, the ‘ID’ field will have to be removed before calculations as the datatype will not work in this algorithms and the ID of the records is not a necessity or a required field for the algorithms.

```
X = traindata.drop(['Type','ID'], axis=1)
y = traindata['Type']
```

The values of each field range by various amounts with some ranging in 10 full values and others only ranging by decimal values.

In preparation for cross-validation, the training data is split so that 30% of it can be used for cross-validation testing and 70% of it is used for training. This was done by:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3,random_state=109)
```

Classifier Selection

The assignment brief states that two classifiers are required, one which should be feature-based and the other should be an end-to-end deep learning model.

For the feature-based classifier, the first option that sprung to mind was to use an SVM. An Support Vector Machine (SVM) is a supervised machine learning algorithm. An SVM can be used or classification or regression purposes, in this case we will be using it for classification. An SVM separates data and tries finds the boundaries of the different data groups. At the boundary, a hyperplane is placed and can be used to classify a set of data. For example, in this case, a records data leads to it’s label prediction by falling within one of the four ‘Types’ or labels. Research was performed in this area and initially I believed I would have to find another classifier type as SVM usually only separate two classes whereas this scenario has four classes. However, further reading lead to the discovery of Support Vector Classifier (SVC). SVM’s follow a straight forward concept and one I understand to some extent, and so I decided to try this SVM SVC method as the feature based classifier.

For the deep learning end-to-end method, an Artificial Neural Network was chosen as it can be used to analyse the data somewhat in a similar simulation to that of a human brain. The similarities in the label sets can be used and analysed through the layers to find the output. Another reason I decided on using a Artificial Neural Network (ANN) as I had wanted to learn more on this technique and develop some experience with it in preparation for my dissertation

project. It also seemed like a good choice of deep learning methods to be used for this scenario, on this scale, and is one of the most popular deep-learning techniques out there. The popularity of ANN's means that if a problem is encountered, there will likely be useful sites or articles found online. This classifier appeared numerous times when researching what deep-learning classifiers are often used for this type of dataset.

A neural network has multiple layers to it, the input layer, numerous hidden layers and the output layer. The input layer inputs the data into the algorithm to process the data further. At this layer, weights can also be added to the different data however in the case of this assignment, the weights were distributed uniformly. The input layer should have the same number of neurons/nodes as there are features in the dataset. The hidden layers are used to process the data input from the previous layer and output a new value based on the activation function it is passed through. There can be numerous hidden layers included in the model although further research on the matter found that using more than 2 layers can overcomplicate the algorithm. The output layer produces the final output for the data. In this case, the output layer will have 4 neurons as there are four possible labels. The output with the largest number for the record input will be used as the predicted label.

Classifier Configuration

Each classifier had its own set of configurations and hyper-parameters, both of which were changed and edited frequently through the production of these algorithms. Below is an explanation of configurations, parameters and architecture of each of the two final models used.

SVM

A linear kernel was used for the SVM with an SVC implemented into the SVM. The C value was best set to 1.

After various testing, it seemed the best option was to stick with these two parameters and let the rest of the SVM settings remain at default as it seemed to produce the best results. SVM takes long to run but resulted in being more accurate in comparison to the ANN results.

The random state used for the SVC was set at 109, this value is used to determine some degree of randomness to the probability estimates.

I believe the reason for the SVM taking such a long time to run is due to how the SVM works, by looking at each possible feature pair for comparison. With the amount of features in the dataset, this is slowing down the run. This can be seen as SVM being unsuitable for this task, however, I plan to stick to this method an analyse the results first.

ANN

Model type used was sequential, so it works in sequence through the layers. Each layer was then added to the model with their own set of configurations.

The input layer contains 1021 neurons as there are 1021 features in the dataset. The input dimensions had to be set at 6000. The initial weights to the data are spread uniformly over the 1021 features so there is no bias at the beginning of the model. ReLU was used as the activation function for the input layer. The Rectified Linear Unit function, when activated takes any negative number as a '0' and positive numbers as a positive activation. All these activations, positive and not, should results in similar patterns being output for similar labels. However, how well this works depends on a large amount of variables.

The two hidden layers both have 500 neurons, numerous tests were executed to get to this number as mentioned in the following experimental section. 500 neurons was the initial value for each hidden layer neuron count as I tried to split the initial 1021 input neurons over the two hidden layers. These two hidden layers also follow the ReLU activation process.

The output layer has 4 neurons, one for each of the possible labels. Softmax was used as the activation function for this final layer as it was the only I could find that would allow for 4 neurons on the final layer. Softmax then produces probabilities of the record being in each label category and so the label with the largest probability will be taken as the label prediction for that record.

Compiling the model itself comes with its own parameters. Loss, optimizer and metrics were the only three parameters used for compiling. Various loss methods were tested however the only method that worked correctly in conjunction with the softmax output layer, was the 'categorical_crossentropy' method. I settled on using the optimizer titled 'adam' as it produced the best results of all methods tried for optimizing. Accuracy is one of the key measurements used to evaluate the algorithm and so the metrics parameter was set to accuracy.

When fitting the model, epoch and batch_size values were added in an attempt to improve results. The final algorithm held 20 epochs and each batch was of size 800. A batch is the number of training samples passed before the algorithms internal parameters are updated. The number of epochs means the number of times the algorithm is run through [2].

The final code for the ANN algorithm is shown below:

```
model = Sequential()
model.add(Dense(1021, input_dim = 6000, init = 'uniform',
activation='relu'))
model.add(Dense(500, activation = 'relu'))
model.add(Dense(500, activation = 'relu'))
model.add(Dense(4, activation = 'softmax'))
```

```
model.compile(loss= 'categorical_crossentropy', optimizer= 'adam' ,
metrics=[ 'accuracy' ])

model.fit(X_train, y_train,
          epochs=20,
          batch_size= 800)
```

Experimentation

Various tests were executed throughout the entirety of this project with testing being executed on each parameter and testing on producing correct results. This section will look more specifically at what was tested for each machine learning model used and what was done as a result of these tests.

SVM

Numerous for loops used within one another to test 3 kernels, a selection of various C values and a selection of various gamma values. This massive test answered which combination of kernel, C value and gamma parameters would produce the best results.

The 3 kernels tested were ‘linear’, ‘rfb’ and ‘sigmoid’. The ‘poly’ kernel was also in an earlier iteration of this test function but took extremely long to execute and I had to cancel the run as it had lasted too long. I believed that ‘poly’ would have been the best kernel choice as we have multiple labels and a large amount of features and so a linear separation will likely not work as well in this scenario as the poly method. This is not known for certain as the poly method could not be tested.

A suitable range of C values were selected, 1, 10 and 100. These values cover a fairly broad range of values but remain within a reasonable range also.

A range of gamma value were also chosen within a suitable range. Enough of a range to check variety but also within a range that a suitable for gamma values of this purpose. The values were 0.0001, 0.01 and 1.0.

The linear kernel produced the best, most accurate results which was not what I was expecting. As a result the linear model was applied as the final model with a C value of 1 and no gamma value as it was not required for the linear model. This provided an accuracy of approximately 80%.

A simple check was executed at the beginning of the testing function to ensure that the labels were given a numerical value and that they were correctly assigned. To check this, the first ten labels of the train set were printed before and after the label encoding, this way I could visually check the conversion.

ANN

Tests were executed for the ANN on various aspects such as the number of neurons at each layer, the type of activation at each layer, the input layer weighting, the loss method, the optimizer type, the number of epochs and the batch size.

The version control on Kaggle proved useful when testing these various factors after numerous tests it was easy to switch back to the earlier iteration of the program with the better results.

The first basic setup used for the ANN used 1021 neurons for the input layer and a uniform weight distribution with ReLU activation. The first hidden layer had a neuron count of 750 and the second hidden layer had a neuron count of 250, both using ReLU activation. The output layer had 1 neuron and a sigmoid activation. The loss method was set to ‘binary_crossentropy’, the optimizer was set to ‘adam’ and the metrics to ‘accuracy’.

This first set-up produced odd results and just listed a number 1 for each record as the output. It was clear at this point that the binary_crossentropy loss method and the current output layer was not suitable for multi-class sorting.

Researching online which parameter set-up is best for a task of this kind led to my discovery of the softmax activation method which was meant to be useful when dealing with multiple classes. However, to use this method, the loss method had to be changed to categorical_crossentropy and the output layer had to increase the number of neurons to 4 (the number of labels). An extra line had to be added to the data preprocessing to convert the numerical labels into ‘dummy labels’ which could be used for the softmax activation method. This then produced an array of length 4 as the output, where the probability of the record belonging to that class/label was stored in the labels spot. The array was set so that the first position hold the probability of the record belonging to ‘A’, the second position for probability of label ‘N’, third position for probability of label ‘O’ and the final position for probability of label ‘~’. Using the softmax method produced useable output and a higher accuracy, bumping up the training and test scores by approximately 5%.

Testing was executed on the activation methods used. Sigmoid, when used on the hidden layers as the activation method produced slightly lower accuracy values than when using the ReLU methods and so the ReLU method remained in use.

Optimizer (where loss = ‘mean_squared_error’)				Optimizer (where loss = ‘categorical_crossentropy’)
Scores	sgd	adam	RMSprop	adam
Training Score	0.30	0.30	0.60	0.99
Test Score	0.30	0.30	0.58	0.50
Label Observations	All data labelled ‘O’	All data labelled ‘O’	All data labelled ‘N’	Mixture of all labels, varies well

This shows that the best option is to go with the adam optimizer with a loss method of categorical_crossentropy as this combination has a high training score and a reasonable test score in comparison to the other test results.

Further testing of the hidden layers was completed, adjusting the neuron amount in each layer. The best results were performed when the number of neurons were spread evenly between the two hidden layers, 500 on one layer and 500 on the other. Although, the difference this change made only improved the accuracy by approximately 4%.

A noticeable issue was present in the output results displayed where the label ‘N’ was significantly more frequent than the other labels, to the point where it looked as though 80-90% of the records were labelled ‘N’. In an attempt to fix this, I added epochs and batch size parameters to the fitting of the model as this would increase the number of run throughs per record and the number of sample used to base the output values off of. I played around with the values of the epochs and batch size until the best results were found. This partly solved the issue of incorrectly assigning ‘N’ to records and their appeared to be a larger variety of labels in the output results along with a small <1% training score increase.

Score	Epochs = 20 Batch Size = 800	Epochs = 10 Batch Size = 200
Training Score	0.99	0.98
Test Score	0.54	0.54

Numerous different optimizer options were tried such as ‘SGD’ with loss as mean_squared_error. Results were less accurate and produce all output as ‘O’ labels. Another optimizer tried was ‘RMSprop’ however, this too produced poor results. The ‘adam’ optimizer seems like the best general optimizer and it is what was used in the final algorithm.

Impact of data pre-processing

The state of the data needs to be set up correctly so that the algorithm inputs correct data. Missing values need to be dealt with, invalid data needs to be sorted and various data types need to be accounted for or encoded. This way there is a set standard for data entering the algorithm so that it can be used effectively and in the same way as the training data.

Without data pre-processing, the algorithm may break with the missing values but more importantly, the algorithm would not work without the label encoding. The label numerical encoding allows for both machine learning methods to account for multiple possible classes as this is not a binary issues but a 4 class issue. The dummy labels are created in the pre-processing of the neural network code and the softmax method in the algorithm relies on this heavily as it would not work without it.

Dimensionality reduction can considerably improve results when applied correctly, however, this was not implemented here for reasons stated in the conclusion. Dimensionality reduction can take a subset of the data and reduce the number of random variables to produce more accurate/precise results.

Classifier Performance Comparison

ANN performed considerably quicker but produced poorer results whereas the SVM took a significantly long time to calculate the oupt and produced a higher accuracy of results.

Result Type	SVM	ANN
Training Accuracy	81%	99%
Test Prediction Accuracy	100%	56%
Time to Run	~2hrs 30mins	~2-5mins

The SVM shows more promising results although the 100% prediction accuracy could mean there is an overfitting issue or just some sort of issue within the algorithm itself. On the other hand, this may not be an issue and the result might be at 100% because these are the test values extracted by the split for this cross-validation and so somehow the labels may already be known or mapped. Both of these functions show decent scores which shows they at least work to some extent.

Classifier Testing on Kaggle

Each classifier was tested on the class kaggle competition. The Neural Network was tested first and resulted in a score of 0.52250. The SVM classifier score is unknown at the time of the completion of this report but should be on the Kaggle competition leaderboards.

Project Language and IDE

I decided on using python for this project as I have some knowledge of this language and find it familiar to work with, it was also suggested to use this language in the assignment brief. Initially, I used Spyder as the IDE but adding some libraries (especially keras and tensorflow) was unnecessarily more complicated and after dealing with long runtimes I instead moved to work on the Kaggle kernel as the IDE where it seems all the libraries are accessible from already and the servers used provided slightly faster runtimes. Used a Windows operated PC to complete the assignment.

Kaggles' implemented version control system was very useful when testing the various parameter changes as if the parameter changes were worsening the algorithm I'd be able to revert to an earlier iteration of the code.

A list of the libraries used is shown in the Appendix of this report.

Main project code is shown in the Appendix in case of the need to reproduce the results.

Conclusion

In conclusion, two machine learning techniques were created with the sole purpose of classifying a set of data as one of four labels. An SVM was used as the feature-based model which proved to produce better results than the ANN but took a painstakingly long time to process. The ANN on the other hand runs considerably faster but produces output with lower accuracy and a larger amount of problems were encountered along the way. The ANN provided a larger number of parameters to tune the model with although no parameters that were tuned increased the quality of results to a satisfactory standard.

Both models work and complete the task even though they can both be massively improved and the accuracy of both are not satisfactory.

Plots weren't produced due to number of features and the required scale of the graph as a result. I attempted to produce a graph during the production of the SVM model however the graph was unreadable. With a more controlled set of data or access to a larger screen, a graph would be a useful way to compare results and performance. I would have also included more experiment tables, however, Kaggle kept disconnecting on my PC and so further tests could not be conducted.

Dimensionality reduction can considerably improve results when applied correctly, however, I tried to implement a dimensionality reduction method known as kernel PCA but I did not understand it fully and due to my lack of understanding could not get a working implementation of the function.

I have learned a lot from this assignment in terms of machine learning model designing and building, although a lot could be improved. Most of the tasks completed I found quite challenging and workload heavy, however I am taking this project as an experience from which I have improved my knowledge of neural networks and machine learning methods. I believe the quality of the work completed for this project is average, but could be greatly improved given greater knowledge on the programming side of the machine learning models and more time. Time was an issue for this assignment as numerous assignments and an exam were due near the date of hand-in.

Bibliography

[1] Brownlee, J. (2019). *What is the Difference Between a Batch and an Epoch in a Neural Network?*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/ [Accessed 24 May 2019].

[2] En.wikipedia.org. (2019). *Kernel method*. [online] Available at: https://en.wikipedia.org/wiki/Kernel_method [Accessed 24 May 2019].

[3] Zerium, A. (2019). *Artificial Neural Networks Explained*. [online] Good Audience. Available at: https://blog.goodaudience.com/artificial-neural-networks-explained-436fcf36e75 [Accessed 24 May 2019].

Appendix

IDE: Kaggle kernel

Language: Python 3.6

Python version and library version are the default versions used on kaggle kernel.

Libraries Used:

- Pandas
    - Dataframe
- Numpy
- Sklearn
    - Model_selection
    - Svm

    - Preprocessing
    - metrics
- Csv
- Keras
    - Models
        - Sequential
    - Layers
        - Dense
    - Utils
        - Np_utils

SVM code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
import csv as csv

#Preprocessing
traindata = pd.read_csv('../input/train_feat.csv')
testdata = pd.read_csv('../input/test_feat.csv')
traindata.fillna(0, inplace=True)
testdata .fillna(0, inplace=True)

X = traindata.drop(['Type','ID'], axis=1)
y = traindata['Type']

X_testdata = testdata.drop('ID', axis=1)

labelencoder = LabelEncoder()
y = labelencoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3,random_state=50)

#Main algorithm
svc = SVC(kernel='linear').fit(X_train, y_train)
scoretrain = svc.score(X_train,y_train)
print(scoretrain)
y_test_pred = svc.predict(X_test)
scoretest  = svc.score(X_test, y_test_pred)
```

```
print(scoretest)

finallabels = []
for i in range(len(y_testdata_pred)):
    if str(y_testdata_pred[i]) == "0":
        finallabels.append("A")
    if str(y_testdata_pred[i]) == "1":
        finallabels.append("N")
    if str(y_testdata_pred[i]) == "2":
        finallabels.append("O")
    if str(y_testdata_pred[i]) == "3":
        finallabels.append("~")

#Saving results to csv file.
#Manually added ID and PredictedClass headers afterwards
lists = zip(ID_testdata, finallabels)
with open("outputSVM.csv","w") as f:
    wr = csv.writer(f)
    for lst in lists:
        wr.writerow(lst)
```

ANN code:

```
import pandas as pd
import numpy as np
import csv

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error

from pandas import DataFrame

import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import np_utils

#preprocessing
traindata = pd.read_csv("../input/train_signal.csv")
testdata = pd.read_csv("../input/test_signal.csv")
traindata.fillna(0, inplace=True)
```

```
testdata .fillna(0, inplace=True)

X = traindata.drop(['Type','ID'], axis=1)
y = traindata['Type']

ID_testdata = testdata['ID']
X_testdata = testdata.drop('ID', axis=1)

labelencoder = LabelEncoder()
y = labelencoder.fit_transform(y)
dummy_y = np_utils.to_categorical(y)

X_train, X_test, y_train, y_test = train_test_split(X, dummy_y,
test_size=0.3)

#Main algorithm
model = Sequential()
model.add(Dense(1021, input_dim = 6000, init = 'uniform',
activation='relu'))
model.add(Dense(500, activation = 'relu'))
model.add(Dense(500, activation = 'relu'))
model.add(Dense(4, activation = 'softmax'))

model.compile(loss= 'categorical_crossentropy', optimizer= 'adam' ,
metrics=[ 'accuracy' ])

model.fit(X_train, y_train,
          epochs=20,
          batch_size= 800)

score1 = model.evaluate(X_train, y_train)
score2 = model.evaluate(X_test, y_test)
print("Train Score: ")
print(score1)
print("Test Score: ")
print(score2)

#cross-validation testing?
pred=model.predict(X_test)
score = np.sqrt(mean_squared_error(y_test,pred))
print("Actual Test Score: ")
print (score)
```

```
#Prediction signal_test data using Neural Network
y_test_nn = model.predict(X_testdata)

#Returns likelihood of test record being in each category
#The following code takes the most likely and assigns it.
finallabels = []
labelval = None
labelind = None

for values in range(len(y_test_nn)):
    labelval = max(y_test_nn[values])
    labelset = y_test_nn[values]
    Labelind = labelset.tolist().index(labelval)
    if str(Labelind) == "0":
        finallabels.append("A")
    if str(Labelind) == "1":
        finallabels.append("N")
    if str(Labelind) == "2":
        finallabels.append("O")
    if str(Labelind) == "3":
        finallabels.append("~")

print(finallabels)
#Saving results to csv file.
#Manually added ID and PredictedClass headers afterwards
lists = zip(ID_testdata, finallabels)
with open("outputSVM.csv","w") as f:
    wr = csv.writer(f)
    for lst in lists:
        wr.writerow(lst)
```