**DEPARTMENT OF ECONOMICS**

UNIVERSITY OF STELLENBOSCH

Correlating Factors of U.S. Presidential Speeches with Stock Market Movements – a Deep Learning Approach

Appendix B

By Pablo Rees

[19119461]

Appendix B formally presented with intention of fulfilment of the requirements for completion of the MCom Economics Programme (full thesis only) at Stellenbosch University.

With supervision from Dawie van Lill

September 01 2022

# Appendix B: Machine Learning Model Specifications and Hyper-Parameters

## B.1. Regression Algorithm Hyper-Parameters

The Stochastic Gradient Descent model uses SciKit Learn’s [SGDRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html#sklearn.linear_model.SGDRegressor) class. I specified the following hyper-parameters:

* loss =' squared\_error'
* max\_iter = 10000
* shuffle = True
* random\_state = 42
* fit\_intercept = False

The Neural Network model uses SciKit Learn’s [MLPRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html) class. I specified the following hyper-parameters:

* hidden\_layer\_sizes = (L\_1, 8, 1)
* activation = 'relu'
* solver = 'sgd'
* max\_iter = 10000
* random\_state = 42

If the number of input variables is above 40 then L\_1 is set to the number of input variables divided by 8 (rounded up to the nearest whole number). If the number of input variables is smaller than 40 then L\_1 is set to 8.

The Multiple Linear Regression model uses SciKit Learn’s [LinearRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html) class. I specified the following hyper-parameters:

* fit\_intercept = False
* n\_jobs = -1

The Gradient Boosting model uses SciKit Learn’s [GradientBoostingRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html) class. I specified the following hyper-parameters:

* random\_state = 42
* loss =' squared\_error'
* learning\_rate = 0.1
* min\_samples\_split = 0.05
* min\_samples\_leaf = 0.02
* max\_depth = 3

## B.2. Classification Algorithm Hyper-Parameters

The Stochastic Gradient Descent model uses SciKit Learn’s [SGDClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html) class. I specified the following hyper-parameters:

* loss =' log’
* random\_state = 42
* shuffle = True
* max\_iter = 10000

The Neural Network model uses SciKit Learn’s [MLPClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html) class. I specified the following hyper-parameters:

* hidden\_layer\_sizes = (L\_1, 8, L\_3)
* activation = 'relu'
* solver = 'sgd'
* max\_iter = 1000
* random\_state = 42

If the number of input variables is above 40 then L\_1 is set to the number of input variables divided by 8 (rounded up to the nearest whole number). If the number of input variables is smaller than 40 then L\_1 is set to 8.

If Binary is True then L\_3 is set to 1 while if Binary is False then L\_3 is set to 7

The Logistic Regression model uses SciKit Learn’s [LogisticRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) class. I specified the following hyper-parameters:

* solver = ‘lbfgs’
* penalty = ‘l2’
* max\_iter = 10000
* random\_state = 42

The Gradient Boosting model uses SciKit Learn’s [GradientBoostingClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html) class. I specified the following hyper-parameters:

* random\_state = 42
* loss =' squared\_error'
* learning\_rate = 0.1
* min\_samples\_split = 0.05
* min\_samples\_leaf = 0.02
* max\_depth = 5

## B.3. Datasets and variables

The full X\_control dataset contains 13 autoregressive variables from the S&P 500 daily closing price. These are:

* DlogDif – the demeaned log difference of the S&P 500 daily closing price (this should be used as a dependent variable only)
* DlogDif\_1 – the first lag of DlogDif
* DlogDif\_2 – the second lag of DlogDif
* absDlogDif – the absolute of DlogDif (this should be used as a dependent variable only)
* absDlogDif\_1 – the first lag of absDlogDif
* logDif – the log difference of the S&P 500 daily closing price (dependent variable only)
* logDif\_date\_resid – the residual of the regression of logDif on the Date variable
* logDif\_date\_resid\_1 – the first lag of logDif\_date\_resid
* blackSwan\_SD3\_1 – a dummy variable that is set to 1 if the first lag of the logDif\_date\_resid variable lies outside 3 standard deviations from its mean and 0 otherwise
* blackSwan\_SD4\_1 – the same as blackSwan\_SD3\_1 but for 4 standard deviations
* blackSwan\_SD5\_1 – the same as blackSwan\_SD3\_1 but for 5 standard deviations
* stdVol\_1DateResid – the first lag of the residual of standardized volume regressed on the Date variable
* pos\_neg\_transform – the first lag of a pseudo-dummy variable which is set to 1 if DlogDif is positive and set to the ratio of the mean of all negative DlogDifs to the mean of all positive DlogDifs

The full X\_meta dataset contains 12 adjacent financial variables that may have correlation with S&P 500 movements. These are:

* BTC\_ld\_1 – the first lag of the log difference of the daily Bitcoin closing price.
* BTC\_dr\_1 – the first lag of the residual of the regression of the Bitcoin closing price on the date.
* Nasdaq\_ld\_1 – the first lag of the log difference of the daily Nasdaq Composite Index closing price.
* Nasdaq\_dr\_1 – the first lag of the residual of the regression of the Nasdaq Composite Index closing price on the date.
* Oil\_ld\_1 – the first lag of the log difference of the daily crude oil closing price.
* Oil\_dr\_1 – the first lag of the residual of the regression of the crude oil closing price on the date.
* SSE\_ld\_1 – the first lag of the log difference of the daily Shanghai Stock Exchange Composite Index closing price.
* SSE\_dr\_1 – the first lag of the residual of the regression of the Shanghai Stock Exchange Composite Index closing price on the date.
* USDX\_ld\_1 – the first lag of the log difference of the daily US Dollar Index closing price.
* USDX\_dr\_1 – the first lag of the residual of the regression of the US Dollar Index closing price on the date.
* VIX\_ld\_1 – the first lag of the log difference of the daily Chicago Board Options Exchange's Volatility Index closing price.
* VIX\_dr\_1 – the first lag of the residual of the regression of the Chicago Board Options Exchange's Volatility Index closing price on the date.

The full X\_test dataset contains 426 NLP variables extracted from the US Presidential Speeches. 6 of these are generated by the VADER and TextBlob sentiment analysis tools. They are:

* VaderNeg – The negativity score generated using the VADER tool.
* VaderNeu – The neutrality score generated using the VADER tool.
* VaderPos – The positivity score generated using the VADER tool.
* VaderComp – The compound score generated using the VADER tool. This is a combination of the negativity, neutrality and positivity scores.
* blobPol – The polarity score generated using the TextBlob tool.
* blobSubj – The subjectivity score generated using the TextBlob tool. The higher the score the more subjective a text is.

200 of these are generated using the Word2Vec text vectorization tool. These variables do not contain specified meaning but are rather the average of final parameters from hidden layers in neural networks performing the pseudo-task of word prediction. Their purpose is to relate words to each other depending on the context of each word within the broader corpus. These are annotated as:

* WV\_0
* WV\_1
* …
* WV\_198
* WV\_199

The final 220 variables are generated using the Doc2Vec text vectorization tool. The Doc2Vec tool works similarly to the Word2Vec tool except that during training a second hidden layer is incorporated. This second layer is persisted across each word in a text in the training corpus and its hyper parameters adjusted so as to create a single vector describing the text as a whole. This tool was used to create 2 sets of variables (each one a hidden layer from a pseudo-task). These are a 200 variable set, annotated as:

* DV\_200\_0
* DV\_200\_1
* …
* DV\_200\_198
* DV\_200\_199

and a 20 variable set, annotated as:

* DV\_20\_0
* DV\_20\_1
* …
* DV\_20\_18
* DV\_20\_19

## B.4. Hyper-parameter tuning for TS Classification models

Two datasets and four models were focussed on during the hyper-parameter tuning for in the TS Classification category. The two datasets were the 1950-01-01 AutoPVDBOW dataset and the 1950-01-01 Auto dataset. While the model focus was on neural networks (NN) and gradient boosting (GB) for the AutoPVDBOW dataset and on logistic regressions (Logreg) and stochastic gradient descent (SGD) for the Auto dataset. Only models that yielded improvements in some score or another are included here. No regression hyper-parameter tuning is reported here because the tuning failed to improve the performance of the regression models.

### B.4.1. Time series classification 1950-01-01 AutoPVDBOW tuning

NN\_1:

elif clf\_type == 'clf\_NN':  
 inputVars = len(XVars)  
 if inputVars / 5 < 8:  
 L\_1 = 8  
 else:  
 L\_1 = round(inputVars / 8)  
  
 if binary:  
 L\_3 = 2  
 else:  
 L\_3 = 7  
  
 clf = MLPClassifier(hidden\_layer\_sizes=(L\_1, 8, L\_3), activation='relu', solver='sgd',  
 max\_iter=10000, random\_state=random\_state)

Dataset: Auto

Train Scores: Accuracy: 0.590 Precision: 0.598Recall: 0.794

Test Scores Accuracy: 0.573 Precision: 0.566 Recall: 0.838

Dataset: AutoPVDBOW

Train Scores: Accuracy: 0.590 Precision: 0.592 Recall: 0.708

Test Scores Accuracy: 0.582 Precision: 0.573 Recall: 0.683

NN\_2:

elif clf\_type == 'clf\_NN':  
 inputVars = len(XVars)  
 if inputVars / 5 < 8:  
 L\_1 = inputVars  
 L\_2 = inputVars  
 else:  
 L\_1 = inputVars  
 L\_2 = inputVars  
  
 if binary:  
 L\_3 = 2  
 else:  
 L\_3 = 7  
  
 clf = MLPClassifier(hidden\_layer\_sizes=(L\_1, L\_2, L\_3), activation='relu', solver='sgd',  
 max\_iter=10000, random\_state=random\_state)

Dataset: Auto

Train Scores: Accuracy: 0.59 Precision: 0.598 Recall: 0.795

Test Scores Accuracy: 0.573 Precision: 0.567 Recall: 0.839

Dataset: AutoPVDBOW

Train Scores: Accuracy: 0.591 Precision: 0.598 Recall: 0.682

Test Scores Accuracy: 0.594 Precision: 0.592 Recall: 0.645

NN\_3: L3 = 5

elif clf\_type == 'clf\_NN':  
 inputVars = len(XVars)  
 if inputVars / 5 < 8:  
 L\_1 = 2\*inputVars  
 L\_2 = inputVars  
 else:  
 L\_1 = 2\*inputVars  
 L\_2 = inputVars  
  
 if binary:  
 L\_3 = 5  
 else:  
 L\_3 = 7  
  
 clf = MLPClassifier(hidden\_layer\_sizes=(L\_1, L\_2, L\_3), activation='tanh', solver='sgd',  
 max\_iter=10000, random\_state=random\_state)

Dataset: Auto

Train Scores: Accuracy: 0.586 Precision: 0.601 Recall: 0.598

Test Scores Accuracy: 0.599 Precision: 0.592 Recall: 0.648

Dataset: AutoPVDBOW

Train Scores: Accuracy: 0.594 Precision: 0.606 Recall: 0.664

Test Scores Accuracy: 0.587 Precision: 0.573 Recall: 0.694

NN\_4: solver = ‘adam’

elif clf\_type == 'clf\_NN':  
 inputVars = len(XVars)  
 if inputVars / 5 < 8:  
 L\_1 = 2\*inputVars  
 L\_2 = inputVars  
 else:  
 L\_1 = 2\*inputVars  
 L\_2 = inputVars  
  
 if binary:  
 L\_3 = 2  
 else:  
 L\_3 = 7  
  
 clf = MLPClassifier(hidden\_layer\_sizes=(L\_1, L\_2, L\_3), activation='relu', solver='adam',  
 max\_iter=10000, random\_state=random\_state)

Dataset: Auto

Train Scores: Accuracy: 0.601 Precision: 0.619 Recall: 0.674

Test Scores Accuracy: 0.587 Precision: 0.59 Recall: 0.777

Dataset: AutoPVDBOW

Train Scores: Accuracy: 0.96 Precision: 0.937 Recall: 0.991 - overfit

Test Scores Accuracy: 0.526 Precision: 0.527 Recall: 0.621

NN\_5: solver = adam, early\_stopping = True

elif clf\_type == 'clf\_NN':  
 inputVars = len(XVars)  
 if inputVars / 5 < 8:  
 L\_1 = 2\*inputVars  
 L\_2 = inputVars  
 else:  
 L\_1 = 2\*inputVars  
 L\_2 = inputVars  
  
 if binary:  
 L\_3 = round(inputVars/2)  
 else:  
 L\_3 = 7  
  
 clf = MLPClassifier(hidden\_layer\_sizes=(L\_1,L\_2,L\_3), activation='relu', solver='adam',  
 max\_iter=10000, random\_state=random\_state,early\_stopping=True)

Dataset: Auto

Train Scores: Accuracy: 0.595 Precision: 0.6 Recall: 0.702

Test Scores Accuracy: 0.581 Precision: 0.563 Recall: 0.736

Dataset: AutoPVDBOW

Train Scores: Accuracy: 0.605 Precision: 0.621 Recall: 0.657

Test Scores Accuracy: 0.592 Precision: 0.585 Recall: 0.793

GB\_1:

else : #clf\_type == 'clf\_GradientBoosting'  
 clf = GradientBoostingClassifier(random\_state=random\_state,learning\_rate=0.01,  
 min\_samples\_split=0.025,min\_samples\_leaf=0.01,max\_depth=8)

Dataset: Auto

Train Scores: Accuracy: 0.506 Precision: 0.491 Recall: 0.412

Test Scores Accuracy: 0.513 Precision: 0.500 Recall: 0.416

Dataset: AutoPVDBOW

Train Scores: Accuracy: 0.512 Precision: 0.5 Recall: 0.441

Test Scores Accuracy: 0.511 Precision: 0.498 Recall: 0.397

GB\_2:

else : #clf\_type == 'clf\_GradientBoosting'  
 clf = GradientBoostingClassifier(random\_state=random\_state,learning\_rate=0.0001,  
 min\_samples\_split=0.01,min\_samples\_leaf=0.001,max\_depth=12)

Dataset: Auto

Train Scores: Accuracy: 0.529 Precision: 0.529 Recall: 1.0

Test Scores Accuracy: 0.523 Precision: 0.523 Recall: 1.0

Dataset: AutoPVDBOW

Train Scores: Accuracy: 0.529 Precision: 0.529 Recall: 1.0

Test Scores Accuracy: 0.524 Precision: 0.524 Recall: 1.0

### B.4.2. Time series classification 1950-01-01 Auto tuning

Logreg\_1:

elif clf\_type == 'clf\_logreg':  
 clf = LogisticRegression(solver='lbfgs', penalty='l2', max\_iter=10000,  
 random\_state=random\_state)

Dataset: Auto

Train Scores: Accuracy: 0.589 Precision: 0.598 Recall: 0.665

Test Scores Accuracy: 0.594 Precision: 0.594 Recall: 0.607

Logreg\_2: fit\_intercept = False

elif clf\_type == 'clf\_logreg':  
 clf = LogisticRegression(solver='lbfgs', penalty='l2', max\_iter=10000,  
 random\_state=random\_state,fit\_intercept=False)

Dataset: Auto

Train Scores: Accuracy: 0.591 Precision: 0.599 Recall: 0.667

Test Scores Accuracy: 0.594 Precision: 0.589 Recall: 0.633

Dataset: AutoPVDBOW

Train Scores: Accuracy: 0.594 Precision: 0.599 Recall: 0.679

Test Scores Accuracy: 0.596 Precision: 0.601 Recall: 0.583

Logreg\_3: penalty=elasticnet

elif clf\_type == 'clf\_logreg':  
 clf = LogisticRegression(solver='saga', penalty='elasticnet', max\_iter=10000,l1\_ratio=0.5,  
 random\_state=random\_state,fit\_intercept=False)

Dataset: Auto

Train Scores: Accuracy: 0.591 Precision: 0.599 Recall: 0.666

Test Scores Accuracy: 0.593 Precision: 0.588 Recall: 0.63

Dataset: AutoPVDBOW

Train Scores: Accuracy: 0.594 Precision: 0.599 Recall: 0.68

Test Scores Accuracy: 0.6 Precision: 0.599 Recall: 0.615

Logreg\_4: L1\_ratio = 0.4

elif clf\_type == 'clf\_logreg':  
 clf = LogisticRegression(solver='saga', penalty='elasticnet', max\_iter=10000,l1\_ratio=0.4,  
 random\_state=random\_state,fit\_intercept=False)

Dataset: Auto

Train Scores: Accuracy: 0.591 Precision: 0.599 Recall: 0.666

Test Scores Accuracy: 0.593 Precision: 0.588 Recall: 0.63

Dataset: AutoPVDBOW

Train Scores: Accuracy: 0.594 Precision: 0.599 Recall: 0.679

Test Scores Accuracy: 0.601 Precision: 0.6 Recall: 0.614

SGD\_1:

if clf\_type == 'clf\_SGD':  
 clf = SGDClassifier(random\_state=random\_state, shuffle=True, loss='log', max\_iter=10000)

Classification type: TS\_Classifier:Classification algo: clf\_SGD

Dataset: Auto

Train Scores: Accuracy: 0.575 Precision: 0.597 Recall: 0.965

Test Scores Accuracy: 0.591 Precision: 0.6 Recall: 0.968

SGD\_2: fit\_intercept = False

if clf\_type == 'clf\_SGD':  
 clf = SGDClassifier(random\_state=random\_state, shuffle=True, loss='log', max\_iter=10000, fit\_intercept=False)

Dataset: Auto

Train Scores: Accuracy: 0.59 Precision: 0.609 Recall: 0.694

Test Scores Accuracy: 0.591 Precision: 0.595 Recall: 0.754

Dataset: AutoPVDBOW

Train Scores: Accuracy: 0.577 Precision: 0.601 Recall: 0.689

Test Scores Accuracy: 0.574 Precision: 0.601 Recall: 0.929

SGD\_3: learning\_rate = adaptive

if clf\_type == 'clf\_SGD':  
 clf = SGDClassifier(random\_state=random\_state, shuffle=True, loss='log', max\_iter=10000, fit\_intercept=False,  
 learning\_rate='adaptive', eta0=0.0001)

Dataset: Auto

Train Scores: Accuracy: 0.583 Precision: 0.597 Recall: 0.606

Test Scores Accuracy: 0.595 Precision: 0.596 Recall: 0.6

Dataset: AutoPVDBOW

Train Scores: Accuracy: 0.588 Precision: 0.595 Recall: 0.662

Test Scores Accuracy: 0.595 Precision: 0.59 Recall: 0.631

SGD\_4: n\_iter\_no\_change = 30

if clf\_type == 'clf\_SGD':  
 clf = SGDClassifier(random\_state=random\_state, shuffle=True, loss='log', max\_iter=10000, fit\_intercept=False,  
 learning\_rate='adaptive', eta0=0.0001, n\_iter\_no\_change=30)

Dataset: Auto

Train Scores: Accuracy: 0.587 Precision: 0.597 Recall: 0.635

Test Scores Accuracy: 0.594 Precision: 0.59 Recall: 0.625

Dataset: AutoPVDBOW

Train Scores: Accuracy: 0.59 Precision: 0.596 Recall: 0.671

Test Scores Accuracy: 0.597 Precision: 0.591 Recall: 0.644

SGD\_5: n\_iter\_no\_change = 100

if clf\_type == 'clf\_SGD':  
 clf = SGDClassifier(random\_state=random\_state, shuffle=True, loss='log', max\_iter=10000, fit\_intercept=False,  
 learning\_rate='adaptive', eta0=0.0001, n\_iter\_no\_change=100)

Dataset: Auto

Train Scores: Accuracy: 0.59 Precision: 0.599 Recall: 0.663

Test Scores Accuracy: 0.595 Precision: 0.591 Recall: 0.631

Dataset: AutoPVDBOW

Train Scores: Accuracy: 0.591 Precision: 0.6 Recall: 0.681

Test Scores Accuracy: 0.597 Precision: 0.59 Recall: 0.652

SGD\_5: eta0=0.001

if clf\_type == 'clf\_SGD':  
 clf = SGDClassifier(random\_state=random\_state, shuffle=True, loss='log', max\_iter=10000, fit\_intercept=False,  
 learning\_rate='adaptive', eta0=0.001, n\_iter\_no\_change=100)

Dataset: Auto

Train Scores: Accuracy: 0.591 Precision: 0.599 Recall: 0.666

Test Scores Accuracy: 0.594 Precision: 0.591 Recall: 0.627

Dataset: AutoPVDBOW

Train Scores: Accuracy: 0.594 Precision: 0.599 Recall: 0.679

Test Scores Accuracy: 0.597 Precision: 0.593 Recall: 0.631