Table of Contents

[References 1](#_Toc126569850)

[Introduction 1](#_Toc126569851)

[Classification Averaging Ensemble Preparation 1](#_Toc126569852)

[numpy.sum() 1](#_Toc126569853)

[np.argmax() 2](#_Toc126569854)

[Eliminating Variance with Ensembles 3](#_Toc126569855)

[Stacking to Improve Accuracy 7](#_Toc126569856)

## References

“Better Deep Learning’ by Dr. Jason Brownlee <https://machinelearningmastery.com/better-deep-learning/>

Chapter 19: Reducing Model Variance with Ensemble Learning

Chapter 20: Combine Models from Multiple Runs with Averaging Ensemble

Chapter 25: Learn to Combine Predictions with Stacked Generalization Ensemble

.

## Introduction

Averaging ensembles calculate the prediction by averaging the predictions of multiple models. Averaging ensembles attempt to reduce variance.

Stacking ensembles calculate the prediction by using the outputs of multiple models as inputs. Stacking ensembles attempt to improve accuracy.

## Classification Averaging Ensemble Preparation

The classification averaging algorithms in this document will use numpy.sum() and numpy.argmax() to help find the average classification for each averaging ensemble. This section will explain how each function works.

### numpy.sum()

**numpy.sum()** adds element values across arrays for each index.

Example : numpy.sum()

|  |
| --- |
| import numpy as np  # Pred1 # Pred2  testArray = np.asarray( [[[0.6, 0.4],[0.2, 0.8]], # Predictions model 1  [[0.1, 0.9],[0.8, 0.2]], # Predictions model 2  [[0.9, 0.1],[0.3, 0.7]], # Predictions model 3  [[0.7, 0.3],[0.6, 0.4]]]) # Predictions model 4  summed = np.sum(testArray, axis=0)  print("\nsummed: ")  print(summed) |

The summed output is shown below.

|  |
| --- |
| [[2.3, 1.7], [1.9, 2.1]] |

Here is the math that generates summed elements for prediction 1:

0.6 0.4

0.1 0.9

0.9 0.1

+ 0.7 0.3

[2.3] [1.7]

Exercise (2 marks)

Show the calculations needed to sum elements by index for all models for prediction 2:

|  |
| --- |
| 0.2 + 0.8 +0.3 + 0.6 = 1.9  0.8 + 0.2 + 0.7 + 0.4 = 2.1 |

### np.argmax()

**np.argmax()** obtains the maximum element value for each of the sub-arrays.

Example : numpy.argmax()

This example shows how numpy.argmax() works. Here the output from Example 1 again. This time though, the largest numbers for each prediction are selected.

# Pred1 # Pred2

[[2.3, 1.7], [1.9, 2.1]]

If you add the following output to the end of the code in Example 1 and run the program again;

|  |
| --- |
| result = np.argmax(summed, axis=1)  print("Max element in array: ")  print(result) |

The output becomes:

[0 1]

Exercise (1 mark)

What is the result of the numpy.argmax() function for the following?

# Pred1 # Pred2

[[8.3, 5.7], [8.9, 8.1]]

|  |
| --- |
|  |

Exercise (1 mark)

What is the result of the numpy.argmax() function for the following?

[[2.3, 1.7, 2.2], [1.9, 2.1, 2.4]]

|  |
| --- |
|  |

We will revisit these functions later.

## Eliminating Variance with Ensembles

You will have noticed during the course that many of your networks exhibit significant variance in accuracy from run-to-run. This section demonstrates how model network ensembles can help to reduce the variance of the output.

Example : Model Variance (See Chapter 20 Better Deep Learning)

This example shows the average accuracy and accuracy standard deviation for 11 separate sequential neural network classifier models.

|  |
| --- |
| Average model accuracy: 0.8111688311688312  Accuracy standard deviation: 0.017505029034279744 |

The following code is set up to build multiple sequential networks for classification. The predictions of each network are then averaged using the numpy.sum() and numpy.argmax() functions. For the initial case in Example 3, only 1 sequential network is used to make the prediction. However, results are generated over 11 separate runs to observe the standard deviation for the results.

Note: The output of the sequential network uses **categorical cross entropy** which causes the classification predictions to be presented in a one-hot-encoded manner.

# Two sample one-hot-encoded outputs. Each output has 1 of 3 outcomes:

# [[1, 0, 0],

# [0, 1, 0],

Here is the code to build the network.

|  |
| --- |
| from keras.layers import Dense  from sklearn.metrics import accuracy\_score  from numpy import argmax  from sklearn.datasets import make\_blobs  from keras.models import Sequential  from sklearn.model\_selection import train\_test\_split  import numpy as np  import tensorflow as tf  # fit model on dataset  def fitModel(trainX, trainy):  # define model  model = Sequential()  model.add(Dense(15, input\_dim=2, activation='relu'))  model.add(Dense(3, activation='softmax'))  model.compile(loss='categorical\_crossentropy',  optimizer='adam',  metrics=['accuracy'])  # fit model  model.fit(trainX, trainy, epochs=200, verbose=0)  return model  def getData():  # generate 2d classification dataset  X, y = make\_blobs(n\_samples=500, centers=3, n\_features=2,  cluster\_std=2, random\_state=2)  # split into train and test  trainX, testX, trainy, testy = train\_test\_split(X, y, test\_size=0.7)  # Converts array to matrix of categories.  # [0, 1, 2]  # Becomes:  # [[1, 0, 0],  # [0, 1, 0],  # [0, 0, 1]]  trainy = tf.keras.utils.to\_categorical(trainy)  return trainX, testX, trainy, testy  def buildAndEvaluateIndividualModels():  trainX, testX, trainy, testy = getData()  NUM\_MODELS = 11  yhats = []  scores = []  models = []  print("\n\*\*\*\* Single model results:")  for i in range(0, NUM\_MODELS):  model = fitModel(trainX, trainy)  models.append(model)  predictions = model.predict(testX)  yhats.append(predictions)  # Converts multi-column prediction set back to single column  # so accuracy score can be calculated.  singleColumnPredictions = argmax(predictions, axis=1)  accuracy = accuracy\_score(singleColumnPredictions, testy)  scores.append(accuracy)  print("Single model " + str(i) + " accuracy: " + str(accuracy))  print("Average model accuracy: " + str(np.mean(scores)))  print("Accuracy standard deviation: " + str(np.std(scores)))  return models  models = buildAndEvaluateIndividualModels() |

Now that you have seen varying results, next we will observe how an ensemble helps to stabilize the results.

Example : Ensemble Predictions (See Chapter 20 Better Deep Learning)

This example shows how results from multiple models are combined to make a more stable prediction by vote. Effectively, the code snippet below (which has already been included in Example 3) adds up votes for each prediction across all models.

On average you will notice that the standard deviation of the ensemble voting model over 11 trials is lowered significantly compared with the ensemble model.

|  |
| --- |
| Average model accuracy: 0.8135064935064935  Accuracy standard deviation: 0.006901470262642296 |

Adding this code to the end of Example 3 will enable the building and evaluating of the ensemble.

|  |
| --- |
| # Evaluate ensemble  def buildAndEvaluateEnsemble(models):  scores = []  print("\n\*\*\*\* Ensemble model results: ")  for trial in range(0, 11):  # Generate new test data.  \_, testX, \_, testy = getData()  yhats = []  # Get predictions with pre-built models.  for model in models:  predictions = model.predict(testX)  yhats.append(predictions)  # Sum predictions for all models.  # [[0.2, 0.3, 0.5], [0.3, 0.3, 0.4]...], # Model 1 results  # [0.3, 0.3, 0.4], [0.1, 0.1, 0.8]...], # Model 2 results  # [0.2, 0.2, 0.6], [0.3, 0.3, 0.4]...], # Model 3 results  # Becomes  # [[0.7, 0.8, 1.5],[0.7, 0.7, 1.6]...] # Summed results  summed = np.sum(yhats, axis=0)  # Converts multi-column prediction set back to single column  # so accuracy score can be calculated. For example;  # [[0.7, 0.8, 1.5],[0.7, 0.7, 1.6]...]  # Becomes  # [2, 2,....]  singleColumnPredictions = argmax(summed, axis=1)  accuracy = accuracy\_score(singleColumnPredictions, testy)  scores.append(accuracy)  print("Ensemble model accuracy during trial " + str(trial) +\  ": " + str(accuracy))  print("Average model accuracy: " + str(np.mean(scores)))  print("Accuracy standard deviation: " + str(np.std(scores)))  buildAndEvaluateEnsemble(models) |

Exercise (6 marks)

Replace the getData() function in Example 4 with the following code to load and prepare data from the Iris data set.

|  |
| --- |
| from sklearn.preprocessing import LabelEncoder  import pandas as pd  from tensorflow.keras.utils import to\_categorical  def getData():  PATH = "/Users/pm/Desktop/DayDocs/data/"  df = pd.read\_csv(PATH + 'iris\_old.csv')  df.columns = ['Sepal L', 'Sepal W', 'Petal L', 'Petal W', 'Iris Type']  # Convert text to numeric category.  # 0 is setosa, 1 is versacolor and 2 is virginica  df['y'] = LabelEncoder().fit\_transform(df['Iris Type'])  # Prepare the data.  X = df[['Sepal L', 'Sepal W', 'Petal L', 'Petal W']]  y = df['y']  ROW\_DIM = 0  COL\_DIM = 1  x\_array = X.values  x\_arrayReshaped = x\_array.reshape(x\_array.shape[ROW\_DIM],  x\_array.shape[COL\_DIM])  y\_array = y.values  y\_arrayReshaped = y\_array.reshape(y\_array.shape[ROW\_DIM], 1)  trainX, testX, trainy, testy = train\_test\_split(x\_arrayReshaped,  y\_arrayReshaped,  test\_size=0.33)  trainy = to\_categorical(trainy)  return trainX, testX, trainy, testy |

Also, adjust the fit\_model() function to enable inputs with 4 features and an output value with 3 possible classifications.

|  |
| --- |
| # fit model on dataset  def fitModel(trainX, trainy):  # define model  model = Sequential()  model.add(Dense(15, input\_dim=4, activation='relu'))  model.add(Dense(3, activation='softmax'))  model.compile(loss='categorical\_crossentropy',  optimizer='adam',  metrics=['accuracy'])  # fit model  model.fit(trainX, trainy, epochs=200, verbose=0)  return model |

Explain how the standard deviation is affected by the ensemble of 11 models.

|  |
| --- |
| It sd is quite low at 0.01 |

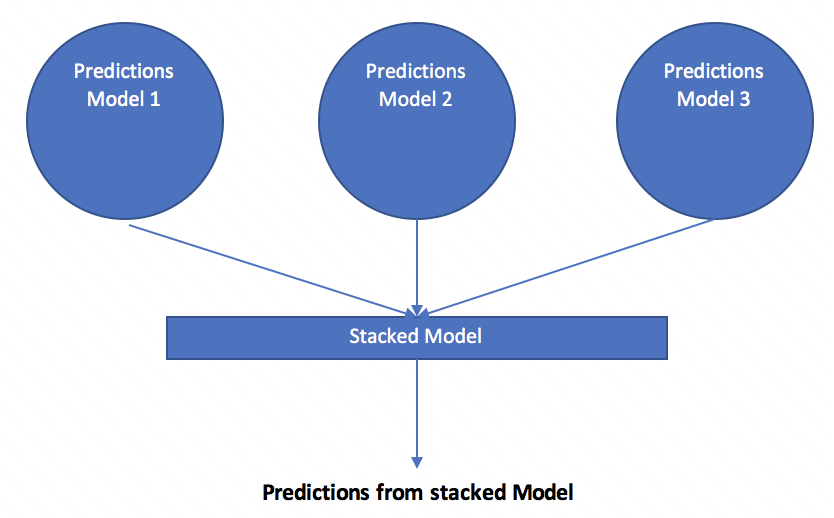
Show your average prediction accuracy and standard deviation for the stand-alone model and for the ensemble.

|  |
| --- |
|  |

## Stacking to Improve Accuracy

Stacking takes the outputs of existing models and uses them as inputs to build a new model. In this way, it is possible to dynamically generate a weighted set of inputs (see Figure 1).

Figure : Stacking Model Built with Inputs from Other Models

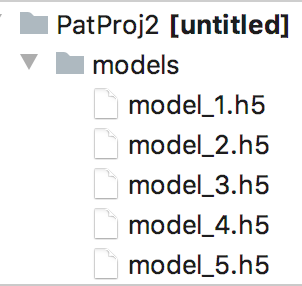


The next two examples show how to implement stacking for a neural network. These examples are discussed at:

<https://machinelearningmastery.com/stacking-ensemble-for-deep-learning-neural-networks/>

Example : Creating Model for Later Use in Stacking Ensemble (See Chapter 25 Better Deep Learning)

This first example creates 5 separate models and saves them as h5 binary files. If you run the code you will notice that it outputs the models in your project directory.



These models will be used in Example 6 for stacking.

|  |
| --- |
| from keras.models import Sequential  from keras.layers import Dense  from os import makedirs  from os import path  from sklearn.model\_selection import train\_test\_split  from sklearn.datasets import make\_blobs  from sklearn.metrics import accuracy\_score  from sklearn.linear\_model import LogisticRegression  from keras.models import load\_model  from tensorflow.keras.utils import to\_categorical  import pandas as pd  import numpy as np  PATH = './models/'  # fit model on dataset  def fit\_model(trainX, trainy):  # define model  model = Sequential()  model.add(Dense(25, input\_dim=2, activation='relu'))  model.add(Dense(3, activation='softmax'))  model.compile(loss='categorical\_crossentropy',  optimizer='adam', metrics=['accuracy'])  # fit model  model.fit(trainX, trainy, epochs=500, verbose=0)  return model  def generateData():  # generate 2d classification dataset  X, y = make\_blobs(n\_samples=800, centers=3,  n\_features=2,  cluster\_std=2, random\_state=2)  # split into train and test  trainX, tempX, trainy, tempy = train\_test\_split(X, y, test\_size=0.6)  testX, valX, testy, valy = train\_test\_split(tempX, tempy, test\_size=0.5)  return trainX, testX, valX, trainy, testy, valy  def generateModels(trainX, trainy):  # create directory for models  if(not path.exists(PATH)):  makedirs('./models')  # fit and save models  numModels = 5  print("\nFitting models with training data.")  for i in range(numModels):  # fit model  model = fit\_model(trainX, trainy)  # save model  filename = PATH + 'model\_' + str(i + 1) + '.h5'  model.save(filename)  print('>Saved %s' % filename)  trainX, testX, valX, trainy, testy, valy = generateData()  # one hot encode output variable  trainy = to\_categorical(trainy)  generateModels(trainX, trainy)  # load models from file  def load\_all\_models(n\_models):  all\_models = list()  for i in range(n\_models):  # define filename for this ensemble  filename = PATH + 'model\_' + str(i + 1) + '.h5'  # load model from file  model = load\_model(filename)  # add to list of models  all\_models.append(model)  print('>loaded %s' % filename)  return all\_models  #trainX, testX, trainy, testy = generateData()  # load all models  numModels = 5  models = load\_all\_models(numModels)  print('Loaded %d models' % len(models))  print("\nEvaluating single models with validation data.")  # evaluate standalone models on test dataset  # individual ANN models are built with one-hot encoded data.  for model in models:  oneHotEncodedY = to\_categorical(valy)  \_, acc = model.evaluate(valX, oneHotEncodedY, verbose=0)  print('Model Accuracy: %.3f' % acc) |

Example : Stacking

This example builds a meta-learner from the models that were created in Example 5. Each of the models that were built in Example 5 show the following accuracy ratings on their own.

|  |
| --- |
| Model Accuracy: 0.818  Model Accuracy: 0.773  Model Accuracy: 0.786  Model Accuracy: 0.777  Model Accuracy: 0.799 |

When predictions from the 5 models are fed as inputs to the stacking model, the stacked model shows a more favourable increase in performance. I have run this code several times and have found the stacked model does not always beat the average of the individual models though there appears a slightly above average performance from the stacked model.

|  |
| --- |
| Stacked Test Accuracy: 0.820 |

Here is the full program.

|  |
| --- |
| from keras.models import Sequential  from keras.layers import Dense  from os import makedirs  from os import path  from sklearn.model\_selection import train\_test\_split  from sklearn.datasets import make\_blobs  from sklearn.metrics import accuracy\_score  from sklearn.linear\_model import LogisticRegression  from keras.models import load\_model  from tensorflow.keras.utils import to\_categorical  import pandas as pd  import numpy as np  PATH = './models/'  # fit model on dataset  def fit\_model(trainX, trainy):  # define model  model = Sequential()  model.add(Dense(25, input\_dim=2, activation='relu'))  model.add(Dense(3, activation='softmax'))  model.compile(loss='categorical\_crossentropy',  optimizer='adam', metrics=['accuracy'])  # fit model  model.fit(trainX, trainy, epochs=500, verbose=0)  return model  def generateData():  # generate 2d classification dataset  X, y = make\_blobs(n\_samples=800, centers=3,  n\_features=2,  cluster\_std=2, random\_state=2)  # split into train and test  trainX, tempX, trainy, tempy = train\_test\_split(X, y, test\_size=0.6)  valX, testX, valY, testY = train\_test\_split(tempX, tempy, test\_size=0.5)  return trainX, valX, testX, trainy, valY, testY  def generateModels(trainX, trainy):  # create directory for models  if(not path.exists(PATH)):  makedirs('./models')  # fit and save models  numModels = 5  print("\nFitting models with training data.")  for i in range(numModels):  # fit model  model = fit\_model(trainX, trainy)  # save model  filename = PATH + 'model\_' + str(i + 1) + '.h5'  model.save(filename)  print('>Saved %s' % filename)  # load models from file  def load\_all\_models(n\_models):  all\_models = list()  for i in range(n\_models):  # define filename for this ensemble  filename = PATH + 'model\_' + str(i + 1) + '.h5'  # load model from file  model = load\_model(filename)  # add to list of models  all\_models.append(model)  print('>loaded %s' % filename)  return all\_models  def evaluateIndividualModels(models, x, y, phase):  individual\_accuracies = []  for model in models:  oneHotEncodedY = to\_categorical(y)  \_, acc = model.evaluate(x, oneHotEncodedY, verbose=0)  print('Model Accuracy: %.3f' % acc)  individual\_accuracies.append(acc)  print("Average individual accuracy during " + phase + " "  + str(np.mean(individual\_accuracies)))  # create stacked model input dataset as outputs from the ensemble  def getStackedData(models, inputX):  stackXdf = None  for model in models:  # make prediction  yhat = model.predict(inputX, verbose=0)  singleModelPredDf = pd.DataFrame(np.row\_stack(yhat))  # Store predictions of all models for 1 sample in each df row.  # Here is 1st row for 5 models with predictions for 3 classes each.  # 5 models x 3 classes = 15 columns.  # 0 1 2 ... 12 13 14  # 0 0.993102 1.106366e-04 0.006788 ... 0.993102 1.106366e-04 0.006788  if stackXdf is None:  stackXdf = singleModelPredDf  else:  numClasses = len(singleModelPredDf.keys())  numStackXCols = len(stackXdf.keys())  # Add new classification columns.  for i in range(0, numClasses):  stackXdf[numStackXCols + i] = stackXdf[i]  return stackXdf  # Make predictions with the stacked model  def stacked\_prediction(models, stackedModel, inputX):  # create dataset using ensemble  stackedX = getStackedData(models, inputX)  # make a prediction  yhat = stackedModel.predict(stackedX)  return yhat  # fit a model based on the outputs from the ensemble models  def fit\_stacked\_model(models, inputX, inputy):  # create dataset using ensemble  stackedX = getStackedData(models, inputX)  # fit standalone model  model = LogisticRegression()  model.fit(stackedX, inputy)  return model  trainX, valX, testX, trainy, valY, testY = generateData()  # Train individual models.  # one hot encode output variable  trainy = to\_categorical(trainy)  generateModels(trainX, trainy)    # load all models  numModels = 5  models = load\_all\_models(numModels)  evaluateIndividualModels(models, testX, testY, 'test')  print('Loaded %d models' % len(models))  print("\nFitting stacked model with test data.")  stackedModel = fit\_stacked\_model(models, testX, testY)  print("\nEvaluating single models with validation data.")  evaluateIndividualModels(models, valX, valY, 'validation')  # evaluate model on test set  print("\nEvaluating stacked model with validation data.")  yhat = stacked\_prediction(models, stackedModel, valX)  acc = accuracy\_score(valY, yhat)  print('Stacked Test Accuracy: %.3f' % acc) |

Exercise (6 marks)

Use the following code to generate five models with the Iris data set.

|  |
| --- |
| from sklearn.preprocessing import LabelEncoder  import pandas as pd  def generateData():  PATH = "/Users/pm/Desktop/DayDocs/data/"  df = pd.read\_csv(PATH + 'iris\_old.csv')  df.columns = ['Sepal L', 'Sepal W', 'Petal L', 'Petal W', 'Iris Type']  # Convert text to numeric category.  # 0 is setosa, 1 is versacolor and 2 is virginica  df['y'] = LabelEncoder().fit\_transform(df['Iris Type'])  # Prepare the data.  X = df[['Sepal L', 'Sepal W', 'Petal L', 'Petal W']]  y = df['y']  # split into train and test  trainX, tempX, trainy, tempy = train\_test\_split(X, y, test\_size=0.6)  testX, valX, testy, valy = train\_test\_split(tempX, tempy, test\_size=0.5)  return trainX, testX, valX, trainy, testy, valy |

Run the code. What adjustment is needed in fit\_model()?

|  |
| --- |
| Input dim needs to be equivalent to the predictor amounts so since we have 4 predictors it needs to be 4 instead of 2 |

Explain the difference in performance between the stacked model and the individual models.

|  |
| --- |
| The the individual models was 0.98  The stacked model was 0.977 this means that it was just as performant as the individual ensemble models |

Exercise (5 marks)

Show a screenshot of the debugger while it shows the dataframe contents. Be sure to show the rows and columns of the stackXdf dataframe at the end of getStackedData() to get this mark.



Show your screenshot here:

|  |
| --- |
|  |

Exercise (4 marks)

Use the terms *ensemble*, *stacked model* to fill in the blanks.

1. A \_\_\_\_\_\_\_ensembles\_\_\_\_\_\_\_\_\_\_\_\_ attempts to reduce variance.
2. A \_\_\_\_\_\_\_\_\_stacked model\_\_\_\_\_\_\_\_\_\_ attempts to improve accuracy.
3. A \_\_\_\_\_\_\_\_\_\_\_stacked model\_\_\_\_\_\_\_\_ builds a model by using the outputs of other models.
4. A \_\_\_\_\_\_\_\_\_ensembles\_\_\_\_\_\_\_\_\_\_ builds a model by averaging the outputs of multiple models.