Libraries and Requirments

In [1]:

```
import pathlib
from typing import Any, Optional
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
import keras
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, InputLayer, Dropout
from kerastuner.tuners import RandomSearch
import tensorflow_probability as tfp
from tensorflow_probability import distributions as tfd
from sklearn.model_selection import train_test_split
from keras import backend as K
from sklearn.preprocessing import OneHotEncoder
!pip install siuba # R syntax for python. Mainly the '>>' operator to work with plotnine
from siuba import *
from plotnine import * # ggplot for python
import itertools
from itertools import chain # for avoiding lists
import math # for sqrt()
import shap

print("Tensorflow version:", tf.__version__)
tf.get_logger().setLevel('ERROR')
print(0)
```

```
Collecting siuba
  Downloading siuba-0.0.25.tar.gz (95 kB)
     |████████████████████████████████| 95 kB 893 kB/s
Requirement already satisfied: pandas>=0.24.0 in /opt/conda/lib/python3.7/site-packages (
from siuba) (1.1.5)
Requirement already satisfied: numpy>=1.12.0 in /opt/conda/lib/python3.7/site-packages (f
rom siuba) (1.19.5)
Requirement already satisfied: SQLAlchemy>=1.2.19 in /opt/conda/lib/python3.7/site-packag
es (from siuba) (1.4.3)
Requirement already satisfied: PyYAML>=3.0.0 in /opt/conda/lib/python3.7/site-packages (f
rom siuba) (5.3.1)
Requirement already satisfied: python-dateutil>=2.7.3 in /opt/conda/lib/python3.7/site-pa
ckages (from pandas>=0.24.0->siuba) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in /opt/conda/lib/python3.7/site-packages (fr
om pandas>=0.24.0->siuba) (2021.1)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.7/site-packages (from p
ython-dateutil>=2.7.3->pandas>=0.24.0->siuba) (1.15.0)
Requirement already satisfied: greenlet!=0.4.17 in /opt/conda/lib/python3.7/site-packages
(from SQLAlchemy>=1.2.19->siuba) (1.0.0)
Requirement already satisfied: importlib-metadata in /opt/conda/lib/python3.7/site-packag
es (from SQLAlchemy>=1.2.19->siuba) (3.4.0)
Requirement already satisfied: typing-extensions>=3.6.4 in /opt/conda/lib/python3.7/site-
packages (from importlib-metadata->SQLAlchemy>=1.2.19->siuba) (3.7.4.3)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.7/site-packages (from
importlib-metadata->SQLAlchemy>=1.2.19->siuba) (3.4.1)
Building wheels for collected packages: siuba
  Building wheel for siuba (setup.py) ... - \ | done
  Created wheel for siuba: filename=siuba-0.0.25-py3-none-any.whl size=112116 sha256=85fa
e495fe85c442501d82f96d511c0799484fe1c682854b39fe5a2d318195d7
  Stored in directory: /root/.cache/pip/wheels/01/8f/63/e72a34fa7ff4166b1bd585418c1849639
188802072db252a81
Successfully built siuba
Installing collected packages: siuba
Successfully installed siuba-0.0.25
Tensorflow version: 2.4.1
```

```
Tensorflow version: 2.4.1
0
```

Helper functions

In [2]:

```
# helper function to plot the history of the dataframe
def plot_history(hist: pd.DataFrame, title="History", file_title="") -> None:
    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.plot(hist['epoch'], hist['accuracy'],
           label='Train Accuracy')
    plt.plot(hist['epoch'], hist['val_accuracy'], alpha=0.5,
           label = 'Validation Accuracy')
    plt.legend()
    plt.title(title)
    plt.savefig(file_title, dpi=500)
    plt.show()
    # plt.figure()
    # plt.xlabel('Epoch')
    # plt.ylabel('Mean Square Error')
    # plt.plot(hist['epoch'], hist['mse'],
    #         label='Train Error')
    # plt.plot(hist['epoch'], hist['val_mse'],
    #         label = 'Val Error')
    # plt.legend()
    # plt.show()
```

In [3]:

```
def plot_confusion_matrix(cm, class_labels=None, file_title=""):
    # From DME labs
    """Plots a confusion matrix using seaborn's heatmap function

    Columns and rows are labelled with the strings provided in class_labels.

    Parameters
    ----------
    cm: array-like
        contains the confusion matrix

    class_labels: array-like, optional
        contains the string labels

    """
    # check whether we have count data or not
    if issubclass(cm.dtype.type, np.integer):
        fmt = 'd'
    else:
        fmt = '.2f'

    # Your code goes here

    if class_labels is not None:
        sns.heatmap(cm, cmap='viridis',xticklabels=class_labels, yticklabels=class_label
s,\
                        annot=True, annot_kws={"fontsize":9},   fmt=fmt)  # controls the disp
lay of the numbers
    else:
        sns.heatmap(cm, annot=True, annot_kws={"fontsize":9},   fmt=fmt)

    plt.ylabel('True label', fontweight='bold')
    plt.xlabel('Predicted label', fontweight='bold')

    # you can change the appearance of the figure with lower-level matplotlib commands
    # here we rotate the labels on the x-axis
    plt.setp(plt.gca().get_xticklabels(), ha="right", rotation_mode="anchor")
    plt.savefig(file_title, dpi=500)
```

In [4]:

```
# Normalization helper function
def norm(x: pd.DataFrame) -> pd.DataFrame:
    return(x - train_stats['mean'] / train_stats['std'])
```

In [5]:

```
def calc_entropy(col):
    entropy = - sum([ p * math.log2(p) for p in col])
    return entropy
```

Data splits

In [6]:

```
raw_wine = pd.read_csv('../input/wine2csv/wine.csv')

# taking a copy
dataset = raw_wine.copy()

# one hot encoding the type
quality = dataset.pop('quality')
dataset['quality_3'] = (quality == 3)*1.0
dataset['quality_4'] = (quality == 4)*1.0
dataset['quality_5'] = (quality == 5)*1.0
dataset['quality_6'] = (quality == 6)*1.0
dataset['quality_7'] = (quality == 7)*1.0
dataset['quality_8'] = (quality == 8)*1.0
dataset['quality_9'] = (quality == 9)*1.0


# setting the X matrix and y vector.
X_full = dataset.loc[ : , dataset.columns != 'type']
y_full = dataset.loc[ : , dataset.columns == 'type']
# pd.to_numeric(y_full.type) # for sparse_categorical_cross_entropy
y_full = OneHotEncoder(sparse=False).fit_transform(dataset[["type"]].values) # for categ
orical_cross_entropy

# Model splits
X_train, X_test, y_train, y_test = train_test_split(X_full, y_full, test_size=0.2, rando
m_state=1903)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, rand
om_state=1903) # 0.25 * (1-0.2) = 0.2

# bug checking splits
# X_train, X_test, y_train, y_test = train_test_split(X_full, y_full, test_size=0.99, ran
dom_state=1903)
# X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.25, random_
state=1903) # 0.25 * (1-0.2) = 0.2

# Normalizing
train_stats = X_train.describe().transpose()
X_train = norm(X_train)
X_val = norm(X_val)
X_test = norm(X_test)

# train_set = pd.concat([X_train, y_train], axis=1)
# test_set  = pd.concat([X_test , y_test ], axis=1)
```

Epoch Levers

In [7]:

```
train_epochs = 2000
search_epochs = 5
ealry_stop_pat = 200
```

Modeling

Hyper parameter tuning Helper Functions

In [8]:

```
def nll(y_true, y_pred):
    """ Negative log likelihood. """

    # keras.losses.binary_crossentropy give the mean
    # over the last axis. we require the sum
    return K.sum(K.categorical_crossentropy(y_true, y_pred), axis=-1)
```

Base

In [9]:

```
# Base model
def model_baseline_hp(hp):
    K.clear_session()
    model = keras.Sequential()
    model.add(keras.layers.Input(shape=18)) # input layer

    # ------------------------------------------------------------
    for i in range(hp.Int('layers', 2, 6)): # number or layers treated as a hyper parame
ter
        model.add(keras.layers.Dense(hp.Int('units_' + str(i),
                                            min_value=32,
                                            max_value=512,
                                            step=128),
                                     activation=hp.Choice('act_' + str(i),
                                                          ['relu', 'sigmoid', 'elu'])))
        # model.add(keras.layers.Dropout(hp.Choice('drop_' + str(i), [0.2, 0.5, 0.7, 0.9]
)))

    # ------------------------------------------------------------
    # model.add(keras.layers.Dense(1, activation='linear')) # output layer for regressio
n
    model.add(keras.layers.Dense(3, activation='softmax')) # output layer for classifica
tion


    # compile the model
    optimizer = keras.optimizers.Adam(hp.Choice('learning_rate', [1e-1, 1e-2, 1e-3, 1e-4
, 1e-5]))
    model.compile(optimizer=optimizer,
                  loss=nll,
                  metrics=['accuracy'])
    return model
```

We illustrate a Bayesian neural network with variational inference, assuming a dataset of features and labels.

It uses the Flipout gradient estimator to minimize the Kullback-Leibler divergence up to a constant, also known as the negative Evidence Lower Bound. It consists of the sum of two terms: the expected negative log-likelihood, which we approximate via Monte Carlo; and the KL divergence, which is added via regularizer terms which are arguments to the layer.

References [1]: Yeming Wen, Paul Vicol, Jimmy Ba, Dustin Tran, and Roger Grosse. Flipout: Efficient Pseudo-Independent Weight Perturbations on Mini-Batches. In International Conference on Learning Representations, 2018. https://arxiv.org/abs/1803.04386

Fliptout

In [10]:

```
# Bayesian model with flipout layer
def model_bayesian_hp_flipout(hp):
    K.clear_session()
    model = keras.Sequential()
    model.add(keras.layers.Input(shape=18)) # input layer
    model.add(tfp.layers.DenseFlipout(hp.Int('flip_units',
                                                  min_value=32,
                                                  max_value=512,
                                                  step=128),
                                           activation=hp.Choice('flip_act', ['relu', 'si
gmoid', 'elu'])))

    # ------------------------------------------------------------------------------
----------
    for i in range(hp.Int('layers', 2, 8)): # number or layers treated as a hyper parame
ter
        model.add(keras.layers.Dense(hp.Int('units_' + str(i),
                                              min_value=32,
                                              max_value=512,
                                              step=128),
                                       activation=hp.Choice('act_' + str(i),
                                                            ['relu', 'sigmoid', 'elu'])))
        # model.add(keras.layers.Dropout(hp.Choice('drop_' + str(i), [0.2, 0.5, 0.7, 0.9]
)))
    # ------------------------------------------------------------------------------
----------

    # model.add(keras.layers.Dense(1, activation='linear')) # output layer for regressio
n
    model.add(tfp.layers.DenseFlipout(3, activation='softmax')) # output layer for class
ification


    # compile the model
    optimizer = keras.optimizers.Adam(hp.Choice('learning_rate', [1e-3, 1e-4, 1e-5, 1e-6
, 1e-7, 1e-8]))
    model.compile(optimizer=optimizer,
                  loss=nll,
                  metrics=['accuracy'])
    return model
```

Variational

In [11]:

```
# Multivariate Normal zero one prior
def prior_trainable(kernel_size: int, bias_size: int, dtype: Any) -> tf.keras.Model:
    n = kernel_size + bias_size
    return tf.keras.Sequential([
        tfp.layers.VariableLayer(n, dtype=dtype),
        tfp.layers.DistributionLambda(lambda t: tfd.Independent(
            tfd.Normal(loc=t, scale=1),
            reinterpreted_batch_ndims=1)),
    ])

# Therefore, theory tells us that the posterior is also Multivariate Normal
def posterior_mean_field(kernel_size: int, bias_size: int, dtype: Any) -> tf.keras.Model
:
    n = kernel_size + bias_size
    c = np.log(np.expm1(1.))

    return tf.keras.Sequential([
        tfp.layers.VariableLayer(2 * n, dtype=dtype),
        tfp.layers.DistributionLambda(lambda t: tfd.Independent(
            tfd.Normal(loc=t[..., :n],
                       scale=1e-5 + tf.nn.softplus(c + t[..., n:])),
```

```
        reinterpreted_batch_ndims=1)),
    ])
```

In [12]:

```
# Bayesian model with variational layer
def model_bayesian_hp_variational(hp):
    K.clear_session()
    model = keras.Sequential()
    model.add(tfp.layers.DenseVariational(hp.Int('var_units',
                                                 min_value=32,
                                                 max_value=512,
                                                 step=128),
                                          activation=hp.Choice('var_act', ['relu', 'sig
moid', 'elu']),
                                          input_shape=[len(X_train.keys())],
                                          make_posterior_fn=posterior_mean_field,
                                          make_prior_fn=prior_trainable,
                                          kl_weight=1/X_train.shape[0]))

    # ---------------------------------------------------------------------------------
----------
    for i in range(hp.Int('layers', 2, 6)): # number or layers treated as a hyper parame
ter
        model.add(keras.layers.Dense(hp.Int('units_' + str(i),
                                            min_value=32,
                                            max_value=512,
                                            step=128),
                                     activation=hp.Choice('act_' + str(i),
                                                          ['relu', 'sigmoid', 'elu'])))
        # model.add(keras.layers.Dropout(hp.Choice('drop_' + str(i), [0.2, 0.5, 0.7, 0.9]
)))
    # ---------------------------------------------------------------------------------
----------

    # model.add(keras.layers.Dense(1, activation='linear')) # output layer for regressio
n
    model.add(keras.layers.Dense(3, activation='softmax')) # output layer for classifica
tion ##


    # compile the model
    optimizer = keras.optimizers.Adam(hp.Choice('learning_rate', [1e-3, 1e-4, 1e-5, 1e-6
, 1e-7]))
    model.compile(optimizer=optimizer,
                  loss=nll,
                  metrics=['accuracy'])
    return model
```

In [13]:

```
# Overwrite the Dropout layer
class MCDropout(Dropout):
    def call(self, inputs):
        return super().call(inputs, training=True)
```

In [14]:

```
# MCdropout layer model
def model_mcdropout_hp(hp):
    K.clear_session()
    model = keras.Sequential()
    model.add(keras.layers.Input(shape=18)) # input layer

    # ------------------------------------------------------------
    for i in range(hp.Int('layers', 2, 6)): # number or layers treated as a hyper parame
ter
        model.add(keras.layers.Dense(hp.Int('units_' + str(i),
                                            min_value=32,
                                            max_value=512,
                                            step=128),
```

```
                                          activation=hp.Choice('act_' + str(i),
                                                               ['relu', 'sigmoid', 'elu'])))
        model.add(MCDropout(hp.Choice('mcdrop_' + str(i), [0.3, 0.5, 0.7, 0.9])))

    # -------------------------------------------------------------
    # model.add(keras.layers.Dense(1, activation='linear')) # output layer for regressio
n
    model.add(keras.layers.Dense(3, activation='softmax')) # output layer for classifica
tion


    # compile the model
    optimizer = keras.optimizers.Adam(hp.Choice('learning_rate', [1e-1, 1e-2, 1e-3, 1e-4
, 1e-5]))
    model.compile(optimizer=optimizer,
                  loss=nll,
                  metrics=['accuracy'])
    return model
```

Base model

In [15]:

```
tuner = RandomSearch(model_baseline_hp,
                     objective='val_accuracy',
                     max_trials=10,
                     executions_per_trial=2,
                     directory='rand_search_outputs',
                     project_name='base_model')
tuner.search_space_summary()
```

```
Search space summary
Default search space size: 6
layers (Int)
{'default': None, 'conditions': [], 'min_value': 2, 'max_value': 6, 'step': 1, 'sampling'
: None}
units_0 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 128, 'samp
ling': None}
act_0 (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'sigmoid', 'elu'], 'ordered': Fa
lse}
units_1 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 128, 'samp
ling': None}
act_1 (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'sigmoid', 'elu'], 'ordered': Fa
lse}
learning_rate (Choice)
{'default': 0.1, 'conditions': [], 'values': [0.1, 0.01, 0.001, 0.0001, 1e-05], 'ordered'
: True}
```

In [16]:

```
tuner.search(X_train, y_train, epochs=search_epochs,
             validation_data=(X_val, y_val))
tuner.results_summary()
```

```
Trial 10 Complete [00h 00m 04s]
val_accuracy: 0.5780612230300903

Best val_accuracy So Far: 0.6836734712123871
Total elapsed time: 00h 00m 52s
Results summary
Results in rand_search_outputs/base_model
Showing 10 best trials
Objective(name='val_accuracy', direction='max')
Trial summary
Hyperparameters:
```

```
Hyperparameters:
layers: 5
units_0: 32
act_0: elu
units_1: 160
act_1: sigmoid
learning_rate: 0.001
units_2: 416
act_2: elu
units_3: 160
act_3: relu
units_4: 416
act_4: sigmoid
units_5: 160
act_5: elu
Score: 0.6836734712123871
Trial summary
Hyperparameters:
layers: 5
units_0: 160
act_0: sigmoid
units_1: 288
act_1: elu
learning_rate: 0.0001
units_2: 288
act_2: relu
units_3: 288
act_3: elu
units_4: 160
act_4: sigmoid
Score: 0.6668367385864258
Trial summary
Hyperparameters:
layers: 6
units_0: 160
act_0: relu
units_1: 32
act_1: sigmoid
learning_rate: 0.001
units_2: 416
act_2: elu
units_3: 416
act_3: elu
units_4: 288
act_4: relu
units_5: 160
act_5: elu
Score: 0.6520408391952515
Trial summary
Hyperparameters:
layers: 5
units_0: 416
act_0: sigmoid
units_1: 160
act_1: relu
learning_rate: 0.0001
units_2: 416
act_2: sigmoid
units_3: 160
act_3: relu
units_4: 160
act_4: elu
units_5: 160
act_5: elu
Score: 0.6418367326259613
Trial summary
Hyperparameters:
layers: 4
units_0: 32
act_0: relu
units_1: 288
act_1: relu
learning_rate: 1e-05
```

```
learning_rate: 1e-05
units_2: 416
act_2: relu
units_3: 32
act_3: relu
units_4: 160
act_4: sigmoid
Score: 0.6265306174755096
Trial summary
Hyperparameters:
layers: 3
units_0: 160
act_0: relu
units_1: 32
act_1: elu
learning_rate: 1e-05
units_2: 288
act_2: elu
units_3: 288
act_3: elu
units_4: 160
act_4: elu
units_5: 288
act_5: elu
Score: 0.5780612230300903
Trial summary
Hyperparameters:
layers: 6
units_0: 32
act_0: elu
units_1: 32
act_1: relu
learning_rate: 1e-05
units_2: 160
act_2: elu
units_3: 160
act_3: sigmoid
units_4: 160
act_4: elu
units_5: 32
act_5: relu
Score: 0.5602040886878967
Trial summary
Hyperparameters:
layers: 4
units_0: 416
act_0: elu
units_1: 416
act_1: sigmoid
learning_rate: 1e-05
units_2: 416
act_2: sigmoid
units_3: 288
act_3: sigmoid
units_4: 288
act_4: relu
units_5: 288
act_5: sigmoid
Score: 0.5494897961616516
Trial summary
Hyperparameters:
layers: 2
units_0: 416
act_0: sigmoid
units_1: 288
act_1: relu
learning_rate: 0.1
Score: 0.359183669090271
Trial summary
Hyperparameters:
layers: 5
units_0: 288
act_0: elu
```

```
units_1: 160
act_1: elu
learning_rate: 0.1
units_2: 32
act_2: relu
units_3: 32
act_3: relu
units_4: 32
act_4: relu
Score: 0.359183669090271
```

In [17]:

```
tuner.get_best_hyperparameters()[0].values
```

Out[17]:

```
{'layers': 5,
 'units_0': 32,
 'act_0': 'elu',
 'units_1': 160,
 'act_1': 'sigmoid',
 'learning_rate': 0.001,
 'units_2': 416,
 'act_2': 'elu',
 'units_3': 160,
 'act_3': 'relu',
 'units_4': 416,
 'act_4': 'sigmoid',
 'units_5': 160,
 'act_5': 'elu'}
```

In [18]:

```
best_base_model = tuner.get_best_models()[0]
```

In [19]:

```
early_stop = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=ealry_stop_pa
t)
```

In [20]:

```
history_base = best_base_model.fit(x=X_train,
                                   y=y_train,
                                   epochs=train_epochs,
                                   batch_size=len(X_train),
                                   verbose=0,
                                   validation_split=0.2,
                                   callbacks=[early_stop])
```

Variational

In [21]:

```
tuner_var = RandomSearch(model_bayesian_hp_variational,
                     objective='val_accuracy',
                     max_trials=10,
                     executions_per_trial=2,
                     directory='rand_search_outputs',
                     project_name='variational_model')
tuner_var.search_space_summary()
```

```
Search space summary
Default search space size: 8
var_units (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 128, 'samp
ling': None}
```

```
var_act (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'sigmoid', 'elu'], 'ordered': Fa
lse}
layers (Int)
{'default': None, 'conditions': [], 'min_value': 2, 'max_value': 6, 'step': 1, 'sampling'
: None}
units_0 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 128, 'samp
ling': None}
act_0 (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'sigmoid', 'elu'], 'ordered': Fa
lse}
units_1 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 128, 'samp
ling': None}
act_1 (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'sigmoid', 'elu'], 'ordered': Fa
lse}
learning_rate (Choice)
{'default': 0.001, 'conditions': [], 'values': [0.001, 0.0001, 1e-05, 1e-06, 1e-07], 'ord
ered': True}
```

In [22]:

```
tuner_var.search(X_train, y_train, epochs=search_epochs,
             validation_data=(X_val, y_val))
tuner_var.results_summary()
```

```
Trial 10 Complete [00h 00m 06s]
val_accuracy: 0.36836734414100647

Best val_accuracy So Far: 0.36836734414100647
Total elapsed time: 00h 01m 07s
Results summary
Results in rand_search_outputs/variational_model
Showing 10 best trials
Objective(name='val_accuracy', direction='max')
Trial summary
Hyperparameters:
var_units: 288
var_act: relu
layers: 2
units_0: 160
act_0: relu
units_1: 416
act_1: relu
learning_rate: 1e-06
units_2: 416
act_2: relu
units_3: 416
act_3: elu
units_4: 160
act_4: elu
Score: 0.36836734414100647
Trial summary
Hyperparameters:
var_units: 288
var_act: relu
layers: 2
units_0: 160
act_0: elu
units_1: 160
act_1: elu
learning_rate: 1e-06
units_2: 32
act_2: sigmoid
units_3: 288
act_3: relu
units_4: 32
act_4: elu
Score: 0.3602040857076645
Trial summary
```

```
Hyperparameters:
var_units: 288
var_act: elu
layers: 4
units_0: 32
act_0: sigmoid
units_1: 160
act_1: elu
learning_rate: 0.0001
units_2: 160
act_2: sigmoid
units_3: 288
act_3: sigmoid
Score: 0.35969386994838715
Trial summary
Hyperparameters:
var_units: 288
var_act: sigmoid
layers: 5
units_0: 160
act_0: elu
units_1: 416
act_1: sigmoid
learning_rate: 0.001
units_2: 160
act_2: sigmoid
units_3: 288
act_3: sigmoid
units_4: 32
act_4: relu
Score: 0.359183669090271
Trial summary
Hyperparameters:
var_units: 416
var_act: relu
layers: 3
units_0: 416
act_0: relu
units_1: 160
act_1: sigmoid
learning_rate: 1e-05
units_2: 416
act_2: elu
units_3: 416
act_3: elu
units_4: 288
act_4: elu
Score: 0.35561223328113556
Trial summary
Hyperparameters:
var_units: 288
var_act: elu
layers: 5
units_0: 416
act_0: sigmoid
units_1: 288
act_1: elu
learning_rate: 1e-05
units_2: 416
act_2: relu
units_3: 160
act_3: relu
units_4: 416
act_4: sigmoid
Score: 0.35153061151504517
Trial summary
Hyperparameters:
var_units: 288
var_act: elu
layers: 2
units_0: 416
act_0: sigmoid
```

```
units_1: 288
act_1: elu
learning_rate: 1e-05
units_2: 160
act_2: sigmoid
units_3: 288
act_3: elu
units_4: 416
act_4: sigmoid
Score: 0.3443877547979355
Trial summary
Hyperparameters:
var_units: 416
var_act: sigmoid
layers: 5
units_0: 416
act_0: sigmoid
units_1: 288
act_1: sigmoid
learning_rate: 1e-06
units_2: 416
act_2: sigmoid
units_3: 160
act_3: sigmoid
units_4: 416
act_4: elu
Score: 0.3443877547979355
Trial summary
Hyperparameters:
var_units: 32
var_act: relu
layers: 4
units_0: 416
act_0: elu
units_1: 32
act_1: relu
learning_rate: 1e-06
units_2: 32
act_2: elu
units_3: 32
act_3: elu
units_4: 288
act_4: relu
Score: 0.3285714238882065
Trial summary
Hyperparameters:
var_units: 32
var_act: elu
layers: 4
units_0: 416
act_0: elu
units_1: 160
act_1: sigmoid
learning_rate: 1e-06
units_2: 32
act_2: relu
units_3: 32
act_3: relu
Score: 0.3214285671710968
```

In [23]:

```
tuner_var.get_best_hyperparameters()[0].values
```

Out[23]:

```
{'var_units': 288,
 'var_act': 'relu',
 'layers': 2,
 'units_0': 160,
 'act_0': 'relu',
 'units_1': 416,
```

```
'act_1': 'relu',
 'learning_rate': 1e-06,
 'units_2': 416,
 'act_2': 'relu',
 'units_3': 416,
 'act_3': 'elu',
 'units_4': 160,
 'act_4': 'elu'}
```

In [24]:

```
best_var_model = tuner_var.get_best_models()[0]
```

In [25]:

```
history_var = best_var_model.fit(x=X_train,
                                 y=y_train,
                                 epochs=train_epochs,
                                 batch_size=32,
                                 verbose=0,
                                 validation_split=0.2,
                                 callbacks=[early_stop])
```

Flipout

In [26]:

```
tuner_flip = RandomSearch(model_bayesian_hp_flipout,
                     objective='val_accuracy',
                     max_trials=10,
                     executions_per_trial=2,
                     directory='rand_search_outputs',
                     project_name='fliptout_model')
tuner_flip.search_space_summary()
```

```
Search space summary
Default search space size: 8
flip_units (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 128, 'samp
ling': None}
flip_act (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'sigmoid', 'elu'], 'ordered': Fa
lse}
layers (Int)
{'default': None, 'conditions': [], 'min_value': 2, 'max_value': 8, 'step': 1, 'sampling'
: None}
units_0 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 128, 'samp
ling': None}
act_0 (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'sigmoid', 'elu'], 'ordered': Fa
lse}
units_1 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 128, 'samp
ling': None}
act_1 (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'sigmoid', 'elu'], 'ordered': Fa
lse}
learning_rate (Choice)
{'default': 0.001, 'conditions': [], 'values': [0.001, 0.0001, 1e-05, 1e-06, 1e-07, 1e-08
], 'ordered': True}
```

```
`layer.add_variable` is deprecated and will be removed in a future version. Please use `l
ayer.add_weight` method instead.
```

In [27]:

```
tuner_flip.search(X_train, y_train, epochs=search_epochs,
```

```
                validation_data=(X_val, y_val))
tuner_flip.results_summary()
```

```
Trial 10 Complete [00h 00m 07s]
val_accuracy: 0.3602040708065033

Best val_accuracy So Far: 0.3857142776250839
Total elapsed time: 00h 01m 19s
Results summary
Results in rand_search_outputs/fliptout_model
Showing 10 best trials
Objective(name='val_accuracy', direction='max')
Trial summary
Hyperparameters:
flip_units: 288
flip_act: sigmoid
layers: 2
units_0: 288
act_0: elu
units_1: 416
act_1: sigmoid
learning_rate: 1e-05
units_2: 288
act_2: sigmoid
units_3: 32
act_3: sigmoid
units_4: 160
act_4: sigmoid
units_5: 416
act_5: sigmoid
units_6: 32
act_6: sigmoid
units_7: 416
act_7: relu
Score: 0.3857142776250839
Trial summary
Hyperparameters:
flip_units: 160
flip_act: sigmoid
layers: 6
units_0: 160
act_0: relu
units_1: 288
act_1: relu
learning_rate: 1e-06
units_2: 416
act_2: relu
units_3: 160
act_3: elu
units_4: 32
act_4: relu
units_5: 32
act_5: relu
Score: 0.3770408183336258
Trial summary
Hyperparameters:
flip_units: 32
flip_act: elu
layers: 3
units_0: 32
act_0: relu
units_1: 288
act_1: sigmoid
learning_rate: 1e-06
units_2: 288
act_2: elu
units_3: 416
act_3: relu
units_4: 160
act_4: relu
units_5: 416
act_5: sigmoid
```

```
units_6: 32
act_6: relu
units_7: 160
act_7: elu
Score: 0.3663265258073807
Trial summary
Hyperparameters:
flip_units: 288
flip_act: sigmoid
layers: 4
units_0: 416
act_0: elu
units_1: 160
act_1: elu
learning_rate: 1e-07
units_2: 32
act_2: relu
units_3: 32
act_3: relu
Score: 0.3632653057575226
Trial summary
Hyperparameters:
flip_units: 288
flip_act: relu
layers: 2
units_0: 32
act_0: sigmoid
units_1: 416
act_1: elu
learning_rate: 1e-05
units_2: 32
act_2: sigmoid
units_3: 160
act_3: sigmoid
units_4: 160
act_4: elu
units_5: 416
act_5: sigmoid
units_6: 416
act_6: relu
units_7: 416
act_7: sigmoid
Score: 0.3602040708065033
Trial summary
Hyperparameters:
flip_units: 32
flip_act: sigmoid
layers: 3
units_0: 160
act_0: elu
units_1: 288
act_1: elu
learning_rate: 1e-06
units_2: 32
act_2: sigmoid
units_3: 32
act_3: relu
units_4: 32
act_4: elu
units_5: 32
act_5: sigmoid
units_6: 416
act_6: relu
units_7: 416
act_7: sigmoid
Score: 0.35663264989852905
Trial summary
Hyperparameters:
flip_units: 416
flip_act: elu
layers: 7
units_0: 32
```

```
act_0: sigmoid
units_1: 416
act_1: sigmoid
learning_rate: 1e-08
units_2: 32
act_2: elu
units_3: 160
act_3: relu
units_4: 32
act_4: elu
units_5: 160
act_5: elu
units_6: 416
act_6: sigmoid
units_7: 288
act_7: relu
Score: 0.3561224490404129
Trial summary
Hyperparameters:
flip_units: 416
flip_act: elu
layers: 8
units_0: 288
act_0: sigmoid
units_1: 160
act_1: sigmoid
learning_rate: 1e-05
units_2: 288
act_2: relu
units_3: 416
act_3: relu
units_4: 32
act_4: sigmoid
units_5: 416
act_5: relu
units_6: 32
act_6: relu
units_7: 32
act_7: relu
Score: 0.3530612289905548
Trial summary
Hyperparameters:
flip_units: 288
flip_act: relu
layers: 4
units_0: 160
act_0: sigmoid
units_1: 288
act_1: elu
learning_rate: 1e-07
units_2: 160
act_2: relu
units_3: 288
act_3: elu
units_4: 416
act_4: relu
units_5: 160
act_5: elu
Score: 0.3392857164144516
Trial summary
Hyperparameters:
flip_units: 416
flip_act: elu
layers: 4
units_0: 416
act_0: elu
units_1: 32
act_1: elu
learning_rate: 1e-07
units_2: 160
act_2: elu
units_3: 32
```

```
act_3: elu
units_4: 288
act_4: elu
units_5: 160
act_5: relu
units_6: 416
act_6: sigmoid
units_7: 160
act_7: relu
Score: 0.3290816396474838
```

In [28]:

```
tuner_flip.get_best_hyperparameters()[0].values
```

Out[28]:

```
{'flip_units': 288,
 'flip_act': 'sigmoid',
 'layers': 2,
 'units_0': 288,
 'act_0': 'elu',
 'units_1': 416,
 'act_1': 'sigmoid',
 'learning_rate': 1e-05,
 'units_2': 288,
 'act_2': 'sigmoid',
 'units_3': 32,
 'act_3': 'sigmoid',
 'units_4': 160,
 'act_4': 'sigmoid',
 'units_5': 416,
 'act_5': 'sigmoid',
 'units_6': 32,
 'act_6': 'sigmoid',
 'units_7': 416,
 'act_7': 'relu'}
```

In [29]:

```
best_flip_model = tuner_flip.get_best_models()[0]
```

```
`layer.add_variable` is deprecated and will be removed in a future version. Please use `l
ayer.add_weight` method instead.
```

In [30]:

```
history_flip = best_flip_model.fit(x=X_train,
                    y=y_train,
                    epochs=train_epochs,
                    batch_size=len(X_train),
                    verbose=0,
                    validation_split=0.2,
                    callbacks=[early_stop])
```

MCDropout

In [31]:

```
tuner_mcdrop = RandomSearch(model_mcdropout_hp,
                     objective='val_accuracy',
                     max_trials=10,
                     executions_per_trial=2,
                     directory='rand_search_outputs',
                     project_name='mcdrop_model')
tuner_mcdrop.search_space_summary()
```

```
Search space summary
Default search space size: 8
```

```
layers (Int)
{'default': None, 'conditions': [], 'min_value': 2, 'max_value': 6, 'step': 1, 'sampling'
: None}
units_0 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 128, 'samp
ling': None}
act_0 (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'sigmoid', 'elu'], 'ordered': Fa
lse}
mcdrop_0 (Choice)
{'default': 0.3, 'conditions': [], 'values': [0.3, 0.5, 0.7, 0.9], 'ordered': True}
units_1 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 128, 'samp
ling': None}
act_1 (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'sigmoid', 'elu'], 'ordered': Fa
lse}
mcdrop_1 (Choice)
{'default': 0.3, 'conditions': [], 'values': [0.3, 0.5, 0.7, 0.9], 'ordered': True}
learning_rate (Choice)
{'default': 0.1, 'conditions': [], 'values': [0.1, 0.01, 0.001, 0.0001, 1e-05], 'ordered'
: True}
```

In [32]:

```
tuner_mcdrop.search(X_train, y_train, epochs=search_epochs,
             validation_data=(X_val, y_val))
tuner_mcdrop.results_summary()
```

```
Trial 10 Complete [00h 00m 05s]
val_accuracy: 0.359183669090271

Best val_accuracy So Far: 0.3622449040412903
Total elapsed time: 00h 00m 54s
Results summary
Results in rand_search_outputs/mcdrop_model
Showing 10 best trials
Objective(name='val_accuracy', direction='max')
Trial summary
Hyperparameters:
layers: 3
units_0: 288
act_0: elu
mcdrop_0: 0.9
units_1: 416
act_1: sigmoid
mcdrop_1: 0.7
learning_rate: 0.001
units_2: 32
act_2: relu
mcdrop_2: 0.3
Score: 0.3622449040412903
Trial summary
Hyperparameters:
layers: 4
units_0: 32
act_0: elu
mcdrop_0: 0.3
units_1: 32
act_1: elu
mcdrop_1: 0.9
learning_rate: 0.001
units_2: 160
act_2: elu
mcdrop_2: 0.7
units_3: 160
act_3: relu
mcdrop_3: 0.3
units_4: 416
act_4: sigmoid
mcdrop_4: 0.5
Score: 0.36173468828201294
```

```
Trial summary
Hyperparameters:
layers: 5
units_0: 160
act_0: elu
mcdrop_0: 0.5
units_1: 160
act_1: relu
mcdrop_1: 0.5
learning_rate: 0.1
units_2: 416
act_2: sigmoid
mcdrop_2: 0.3
units_3: 416
act_3: sigmoid
mcdrop_3: 0.7
units_4: 288
act_4: relu
mcdrop_4: 0.5
Score: 0.359183669090271
Trial summary
Hyperparameters:
layers: 5
units_0: 416
act_0: relu
mcdrop_0: 0.7
units_1: 32
act_1: elu
mcdrop_1: 0.7
learning_rate: 1e-05
units_2: 32
act_2: sigmoid
mcdrop_2: 0.3
units_3: 32
act_3: relu
mcdrop_3: 0.3
units_4: 32
act_4: relu
mcdrop_4: 0.3
Score: 0.35765306651592255
Trial summary
Hyperparameters:
layers: 3
units_0: 160
act_0: relu
mcdrop_0: 0.7
units_1: 32
act_1: relu
mcdrop_1: 0.7
learning_rate: 0.0001
units_2: 416
act_2: relu
mcdrop_2: 0.5
units_3: 32
act_3: relu
mcdrop_3: 0.9
units_4: 288
act_4: sigmoid
mcdrop_4: 0.3
Score: 0.3571428507566452
Trial summary
Hyperparameters:
layers: 3
units_0: 160
act_0: relu
mcdrop_0: 0.3
units_1: 32
act_1: relu
mcdrop_1: 0.3
learning_rate: 0.0001
units_2: 160
act_2: relu
```

```
mcdrop_2: 0.9
units_3: 32
act_3: relu
mcdrop_3: 0.3
units_4: 416
act_4: relu
mcdrop_4: 0.3
Score: 0.35561224818229675
Trial summary
Hyperparameters:
layers: 4
units_0: 32
act_0: relu
mcdrop_0: 0.3
units_1: 416
act_1: sigmoid
mcdrop_1: 0.7
learning_rate: 1e-05
units_2: 416
act_2: elu
mcdrop_2: 0.3
units_3: 288
act_3: sigmoid
mcdrop_3: 0.9
units_4: 288
act_4: sigmoid
mcdrop_4: 0.5
Score: 0.3551020324230194
Trial summary
Hyperparameters:
layers: 3
units_0: 416
act_0: elu
mcdrop_0: 0.9
units_1: 160
act_1: sigmoid
mcdrop_1: 0.9
learning_rate: 0.001
units_2: 32
act_2: relu
mcdrop_2: 0.3
units_3: 160
act_3: sigmoid
mcdrop_3: 0.9
units_4: 416
act_4: elu
mcdrop_4: 0.9
Score: 0.3530612289905548
Trial summary
Hyperparameters:
layers: 5
units_0: 288
act_0: sigmoid
mcdrop_0: 0.5
units_1: 32
act_1: relu
mcdrop_1: 0.7
learning_rate: 1e-05
units_2: 160
act_2: relu
mcdrop_2: 0.5
units_3: 32
act_3: elu
mcdrop_3: 0.7
units_4: 416
act_4: relu
mcdrop_4: 0.9
Score: 0.3520408123731613
Trial summary
Hyperparameters:
layers: 4
units_0: 416
```

```
act_0: elu
mcdrop_0: 0.7
units_1: 416
act_1: relu
mcdrop_1: 0.5
learning_rate: 1e-05
units_2: 416
act_2: sigmoid
mcdrop_2: 0.3
units_3: 32
act_3: relu
mcdrop_3: 0.9
units_4: 288
act_4: sigmoid
mcdrop_4: 0.7
Score: 0.3464285731315613
```

In [33]:

```
tuner_mcdrop.get_best_hyperparameters()[0].values
```

Out[33]:

```
{'layers': 3,
 'units_0': 288,
 'act_0': 'elu',
 'mcdrop_0': 0.9,
 'units_1': 416,
 'act_1': 'sigmoid',
 'mcdrop_1': 0.7,
 'learning_rate': 0.001,
 'units_2': 32,
 'act_2': 'relu',
 'mcdrop_2': 0.3}
```

In [34]:

```
best_mcdrop_model = tuner_mcdrop.get_best_models()[0]
```

In [35]:

```
history_mcdrop = best_mcdrop_model.fit(x=X_train,
                                       y=y_train,
                                       epochs=train_epochs,
                                       batch_size=len(X_train),
                                       verbose=0,
                                       validation_split=0.2,
                                       callbacks=[early_stop])
```

Histories

In [36]:

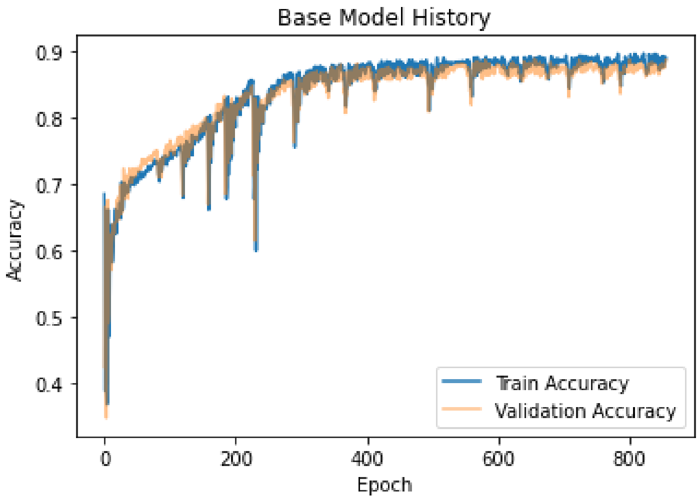
```
best_base_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                608
_________________________________________________________________
dense_1 (Dense)              (None, 160)               5280
_________________________________________________________________
dense_2 (Dense)              (None, 416)               66976
_________________________________________________________________
dense_3 (Dense)              (None, 160)               66720
_________________________________________________________________
dense_4 (Dense)              (None, 416)               66976
_________________________________________________________________
dense_5 (Dense)              (None, 3)                 1251
=================================================================
```

```
Total params: 207,811
Trainable params: 207,811
Non-trainable params: 0
_________________________________________________________________
```

In [37]:

```
hist = pd.DataFrame(history_base.history)
hist['epoch'] = history_base.epoch
plot_history(hist, title='Base Model History', file_title='a.history-base.pdf')
```



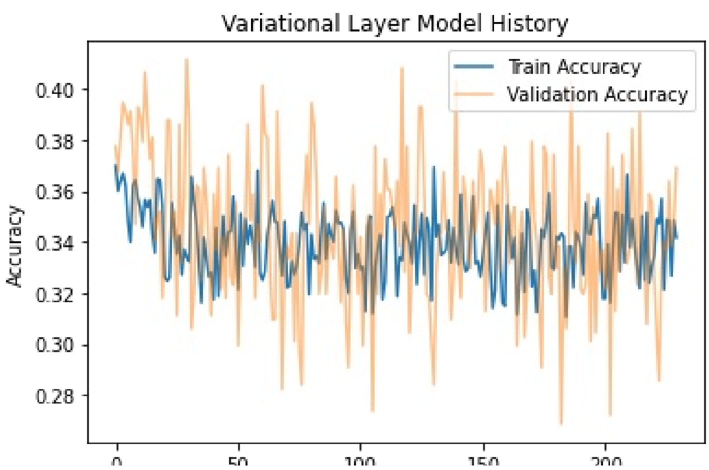
In [38]:

```
best_var_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_variational (DenseVari (None, 288)               16416
_________________________________________________________________
dense (Dense)                (None, 160)               46240
_________________________________________________________________
dense_1 (Dense)              (None, 416)               66976
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 1251
=================================================================
Total params: 130,883
Trainable params: 130,883
Non-trainable params: 0
_________________________________________________________________
```

In [39]:

```
hist = pd.DataFrame(history_var.history)
hist['epoch'] = history_var.epoch
h2 = plot_history(hist, title='Variational Layer Model History', file_title='b.history-v
ar.pdf')
```



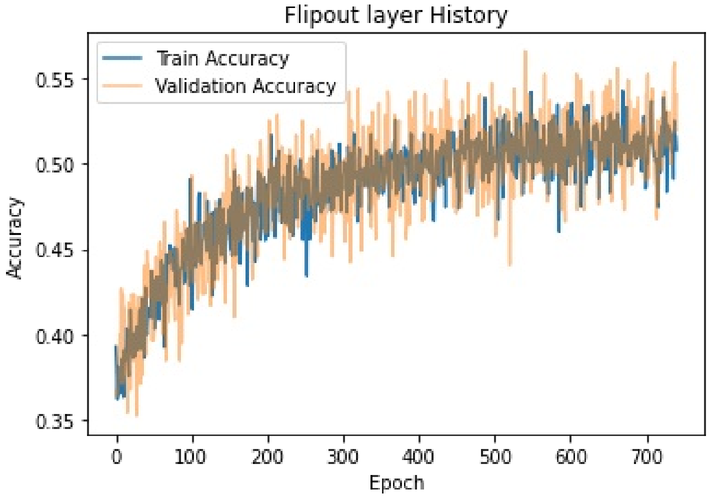
In [40]:

```
best_flip_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_flipout (DenseFlipout) (None, 288)               10656     
_________________________________________________________________
dense (Dense)                (None, 288)               83232     
_________________________________________________________________
dense_1 (Dense)              (None, 416)               120224    
_________________________________________________________________
dense_flipout_1 (DenseFlipou (None, 3)                 2499      
=================================================================
Total params: 216,611
Trainable params: 216,611
Non-trainable params: 0
_________________________________________________________________
```

In [41]:

```
hist_flip = pd.DataFrame(history_flip.history)
hist_flip['epoch'] = history_flip.epoch
h3 = plot_history(hist_flip, title='Flipout layer History', file_title='c.history-flip.p
df')
```



In [42]:

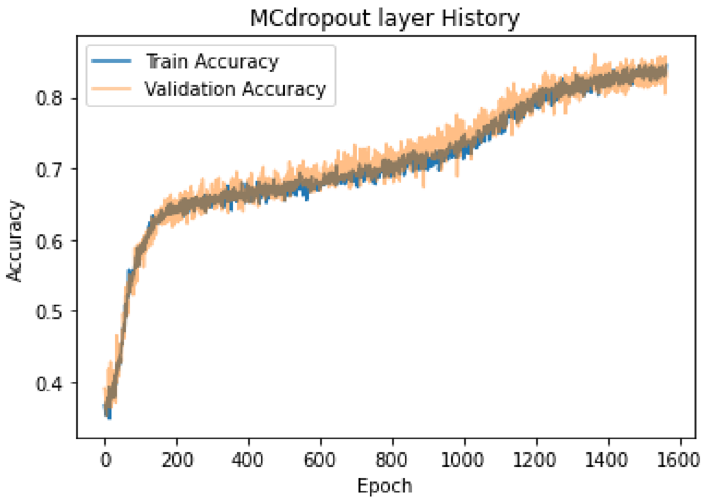
```
best_mcdrop_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 288)               5472      
_________________________________________________________________
mc_dropout (MCDropout)       (None, 288)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 416)               120224    
_________________________________________________________________
mc_dropout_1 (MCDropout)     (None, 416)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 32)                13344     
_________________________________________________________________
mc_dropout_2 (MCDropout)     (None, 32)                0         
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 99        
=================================================================
Total params: 139,139
Trainable params: 139,139
```

```
Non-trainable params: 0
_________________________________________________________________
```

In [43]:

```
hist_mcdrop = pd.DataFrame(history_mcdrop.history)
hist_mcdrop['epoch'] = history_mcdrop.epoch
h4 = plot_history(hist_mcdrop, title='MCdropout layer History', file_title='d.history-mc
drop.pdf')
```

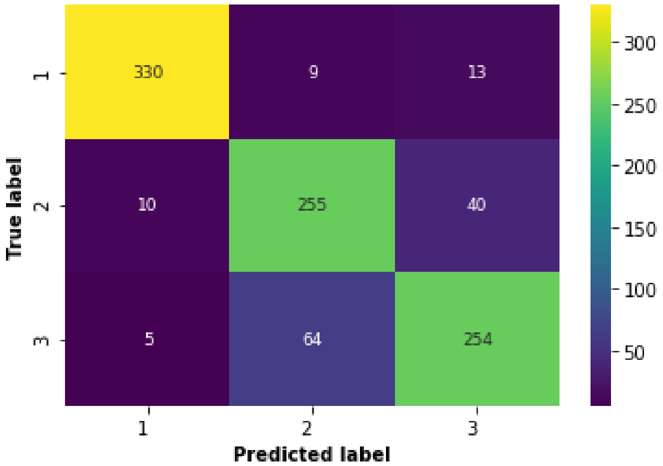


Model Validation

In [44]:

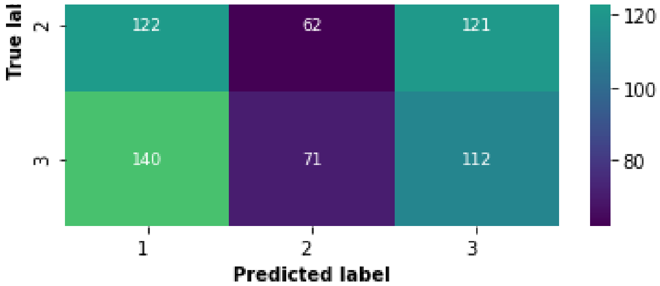
```
from sklearn.metrics import confusion_matrix

p_base = best_base_model.predict(X_val)
cf_mat_base = confusion_matrix(np.argmax(y_val, axis=1),
                 np.argmax(p_base, axis=1))
plot_confusion_matrix(cf_mat_base, class_labels=[1, 2, 3], file_title='e.confmat-base.pd
f')
```



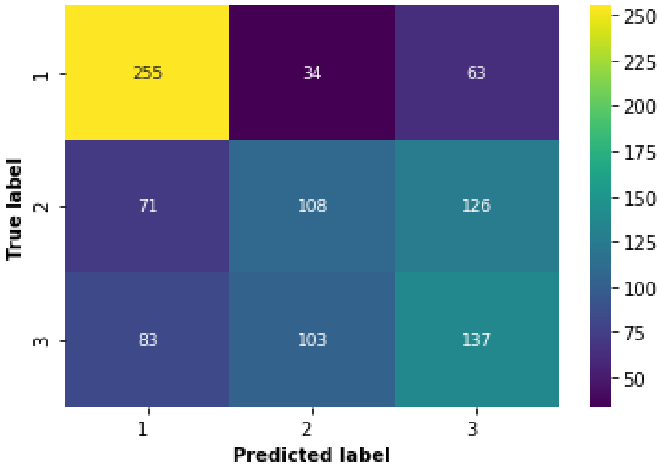
In [45]:

```
p_var = best_var_model.predict(X_val)
cf_mat_var = confusion_matrix(np.argmax(y_val, axis=1),
                 np.argmax(p_var, axis=1))
plot_confusion_matrix(cf_mat_var, class_labels=[1, 2, 3], file_title='f.confmat-var.pdf'
)
```



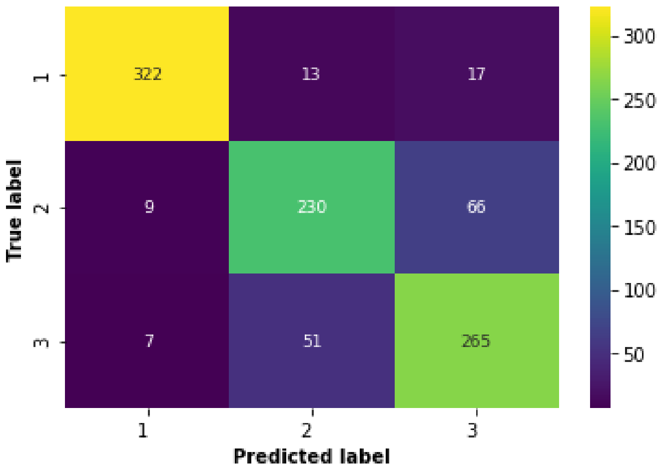
In [46]:

```
p_flip = best_flip_model.predict(X_val)
cf_mat_flip = confusion_matrix(np.argmax(y_val, axis=1),
                  np.argmax(p_flip, axis=1))
plot_confusion_matrix(cf_mat_flip, class_labels=[1, 2, 3], file_title='g.confmat-flip.pd
f')
```



In [47]:

```
p_mcdrop = best_mcdrop_model.predict(X_val)
cf_mat_mcdrop = confusion_matrix(np.argmax(y_val, axis=1),
                                  np.argmax(p_mcdrop, axis=1))
plot_confusion_matrix(cf_mat_mcdrop, class_labels=[1, 2, 3], file_title='h.confmat-mcdro
p.pdf')
```



The traditional NN works better, but within the Bayesian approaches, the MCdropout works best

Model Explanation

In [48]:

```
# Select the best model for the analysis
model = best_mcdrop_model
```

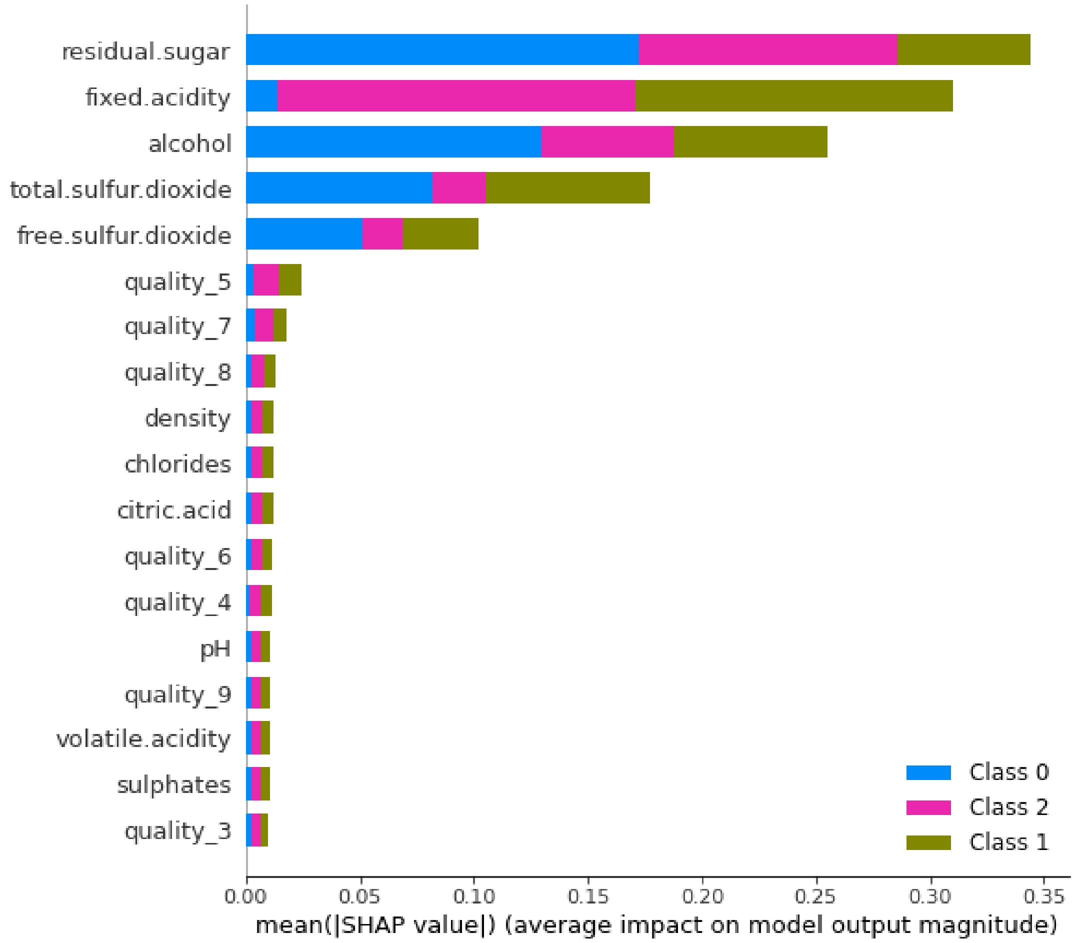
In [49]:

In [49]:

```
# calculate SHAP values
explainer = shap.KernelExplainer(model.predict, X_test)
shap_values = explainer.shap_values(X_test, nsamples=100)
```

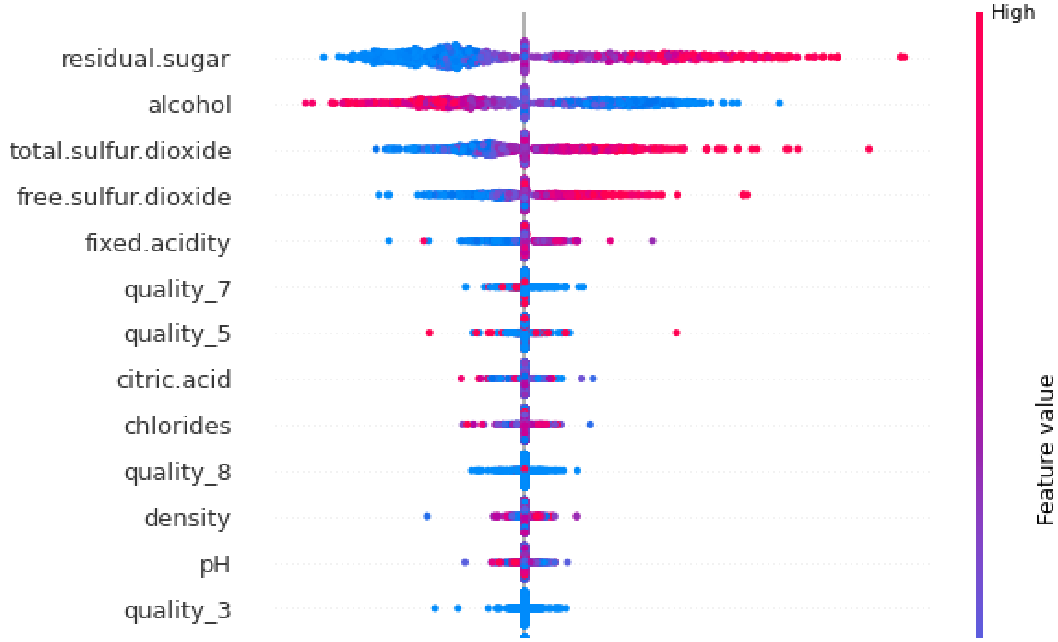
In [50]:

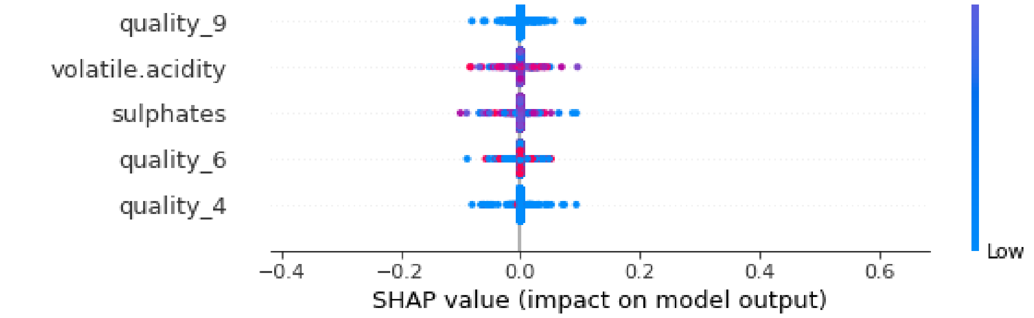
```
# Plot summary
shap.summary_plot(shap_values, X_test, feature_names=X_test.columns, plot_type="bar", sh
ow=False)
plt.savefig('i.shap-summary.pdf', dpi=500)
```



In [51]:

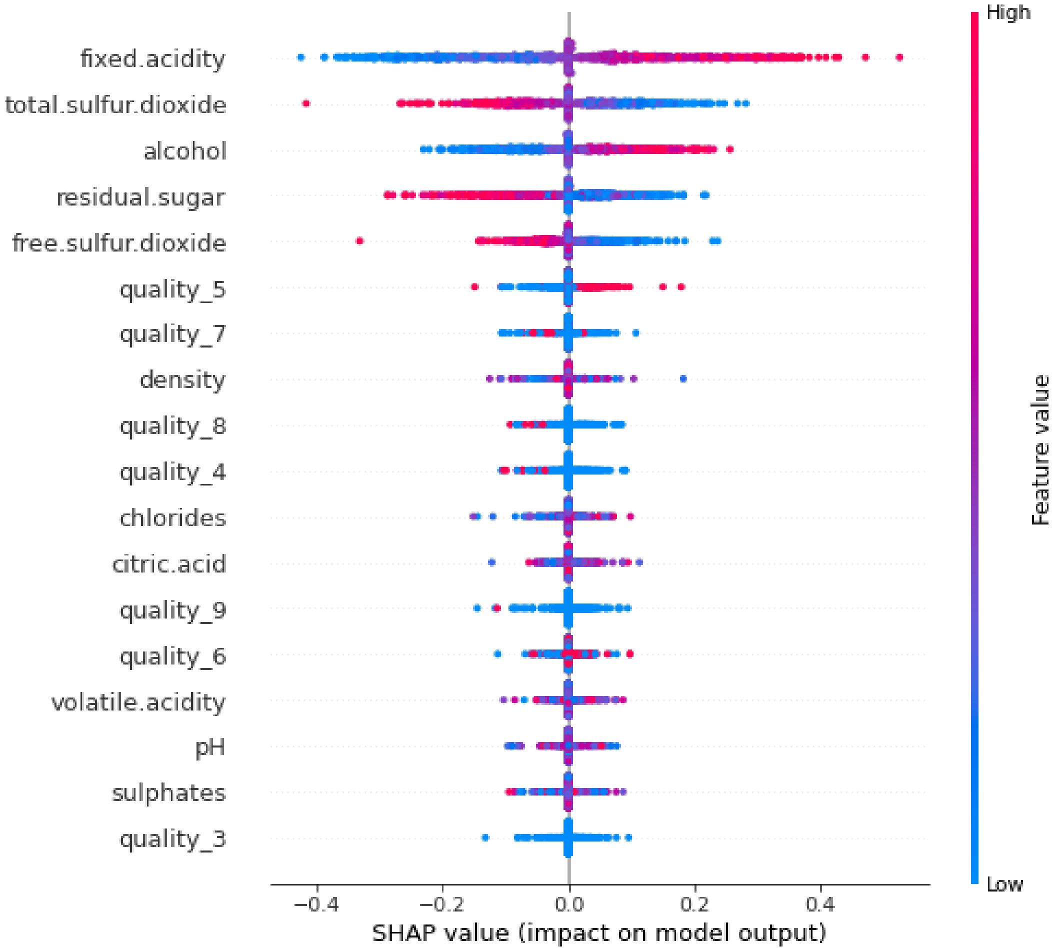
```
# Plot Summary for class 0
shap.summary_plot(shap_values[0], X_test, feature_names=X_test.columns, show=False)
plt.savefig('j.shap-class0.pdf', dpi=500)
```





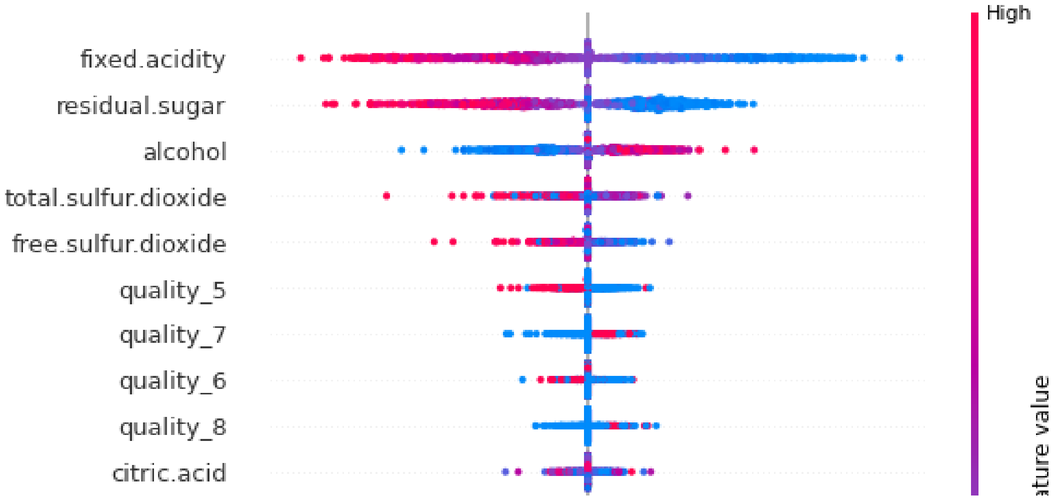
In [52]:

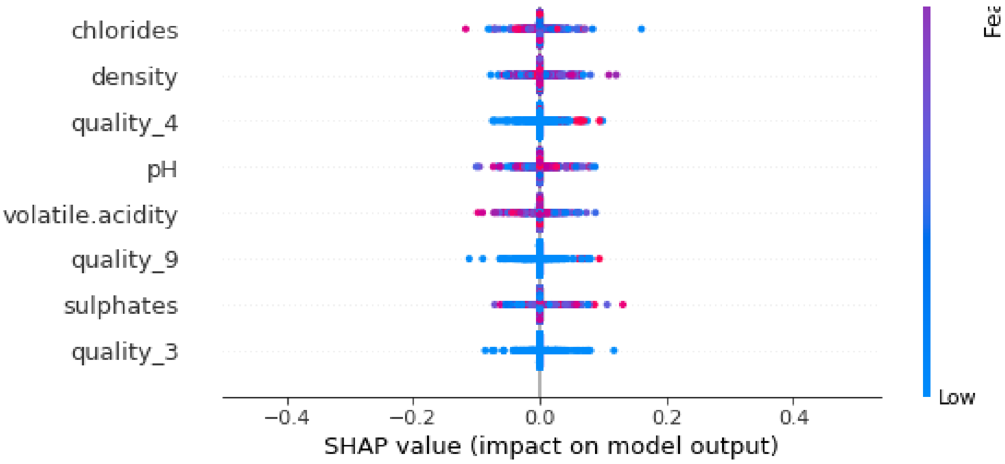
```
# Plot Summary for class 1
shap.summary_plot(shap_values[1], X_test, feature_names=X_test.columns, show=False)
plt.savefig('k.shap-class1.pdf', dpi=500)
```



In [53]:

```
# Plot Summary for class 2
shap.summary_plot(shap_values[2], X_test, feature_names=X_test.columns, show=False)
plt.savefig('l.shap-class2.pdf', dpi=500)
```





MCMC Sampling

https://github.com/zacharyaanglin/ProbabilisticDeepLearningTensorFlow/blob/master/2%20-%20Simple%20linear%20regression%20in%20Keras%20with%20uncertainty.ipynb

https://colab.research.google.com/drive/149Wg0uB0x1ZrPDMme8gyOnlsgHC9JM6M?hl=en&pli=1#scrollTo=Bxhbrf1R1UPE

In [54]:

```
# draw predictions n times
n_samples = 1000
yhats = [model.predict(X_test) for _ in range(n_samples)]
n_set = X_test.shape[0]
```

In [55]:

```
# fix format of output
yhat_samples = pd.DataFrame(list(itertools.chain.from_iterable(yhats)))
```

In [56]:

```
# fix format of output
yhat_samples = []
for i in range(n_samples):
    yhat_samples.extend(yhats[i])
yhat_samples = pd.DataFrame(yhat_samples)
```

In [57]:

```
# create id columns for validation number and sample number
id_col = list(range(n_set)) # validation set number
samples_col = np.repeat(np.arange(n_samples), n_set, axis=0) # sample number
```

In [58]:

```
# add extra columns and format
yhat_samples["id"] = id_col*n_samples
id_col = yhat_samples.pop("id")
yhat_samples.insert(0, "val_id", id_col)
yhat_samples.insert(1, "sam_id", samples_col)
yhat_samples.insert(5, "true_1", pd.DataFrame(list(y_val)*n_samples)[0])
yhat_samples.insert(6, "true_2", pd.DataFrame(list(y_val)*n_samples)[1])
yhat_samples.insert(7, "true_3", pd.DataFrame(list(y_val)*n_samples)[2])
yhat_samples.columns = ['val_id', 'sam_id', 'pred_1', 'pred_2', 'pred_3', 'true_1', 'true
_2', 'true_3'] # rename column names
```

In [59]:

```
# yhat_samples['Entropy'] = -(values*np.log(values))
entropies = pd.DataFrame(yhat_samples.groupby('val_id')[['pred_1', 'pred_2', 'pred_3', '
```

```
true_1', 'true_2', 'true_3']].mean())
shannon = entropies.loc[:,'pred_1':'pred_3'].apply(calc_entropy,axis=1)
entropies['entropy'] = shannon
top_10 = entropies.sort_values("entropy", ascending=False).head(5)
top_10_inx = top_10.index
top_10
```

Out[59]:

val_id	pred_1	pred_2	pred_3	true_1	true_2	true_3	entropy
47	0.327196	0.350532	0.322272	1.0	0.0	0.0	1.583983
297	0.315351	0.335811	0.348838	0.0	0.0	1.0	1.583724
859	0.362033	0.290577	0.347391	1.0	0.0	0.0	1.578670
530	0.319214	0.288024	0.392763	1.0	0.0	0.0	1.572640
744	0.292019	0.402495	0.305486	1.0	0.0	0.0	1.569677

In [60]:

```
confidence = yhat_samples >> group_by(_.val_id) >> mutate(avg_1 = _.pred_1.mean(),
                                                           avg_2 = _.pred_2.mean(),
                                                           avg_3 = _.pred_3.mean(),
                                                           std_1 = _.pred_1.std(),
                                                           std_2 = _.pred_2.std(),
                                                           std_3 = _.pred_3.std(),
                                                           ci_95_l_1 = _.avg_1-1.96*_.pr
ed_1.std()/math.sqrt(n_samples),
                                                           ci_95_h_1 = _.avg_1+1.96*_.pr
ed_2.std()/math.sqrt(n_samples),
                                                           ci_95_l_2 = _.avg_2-1.96*_.pr
ed_3.std()/math.sqrt(n_samples),
                                                           ci_95_h_2 = _.avg_2+1.96*_.pr
ed_1.std()/math.sqrt(n_samples),
                                                           ci_95_l_3 = _.avg_3-1.96*_.pr
ed_2.std()/math.sqrt(n_samples),
                                                           ci_95_h_3 = _.avg_3+1.96*_.pr
ed_3.std()/math.sqrt(n_samples),
                                                           diff_ci_1 = _.ci_95_h_1 - _.c
i_95_l_1,
                                                           diff_ci_2 = _.ci_95_h_2 - _.c
i_95_l_2,
                                                           diff_ci_3 = _.ci_95_h_3 - _.c
i_95_l_3) >> filter(_.sam_id==0)
```

In [61]:

```
pred_ci = confidence[['true_1', 'true_2', 'true_3',
                      'avg_1', 'avg_2', 'avg_3',
                      'ci_95_l_1', 'ci_95_h_1',
                      'ci_95_l_2', 'ci_95_h_2',
                      'ci_95_l_3', 'ci_95_h_3',
                      'std_1', 'std_2', 'std_3',
                      'diff_ci_1', 'diff_ci_2', 'diff_ci_3']]
pred_ci >> arrange(-_.diff_ci_2)
```

Out[61]:

(grouped data frame)

	val_id	sam_id	pred_1	pred_2	pred_3	true_1	true_2	true_3	avg_1	avg_2	...	std_3	ci_95_l_1	ci_9
117	117	0	0.414007	5.344124e-02	5.325520e-01	0.0	1.0	0.0	0.498555	0.048142	...	0.234040	0.483351	0.5
275	275	0	0.066162	1.561439e-01	7.776940e-01	1.0	0.0	0.0	0.536481	0.052153	...	0.227396	0.521237	0.5
804	804	0	0.850973	8.459583e-04	1.481809e-01	0.0	0.0	1.0	0.694974	0.025653	...	0.222004	0.680263	0.6

	val_id	sam_id	pred_1	pred_2	pred_3	true_1	true_2	true_3	avg_1	avg_2	...	std_3	ci_95_l_1	ci_9
112	112	0	0.161654	7.679272e-03	8.306664e-01	0.0	0.0	1.0	0.410879	0.064707	...	0.220548	0.396526	0.4
542	542	0	0.951809	9.874147e-03	3.831648e-02	0.0	0.0	1.0	0.692612	0.038017	...	0.216240	0.678105	0.6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
721	721	0	0.999705	1.382324e-04	1.569449e-04	0.0	0.0	1.0	0.996084	0.002551	...	0.008464	0.995041	0.9
186	186	0	1.000000	5.135615e-08	1.495656e-08	0.0	0.0	1.0	0.995403	0.003114	...	0.007003	0.994282	0.9
866	866	0	0.999999	6.143734e-07	4.576464e-07	0.0	1.0	0.0	0.995884	0.002231	...	0.008469	0.994993	0.9
829	829	0	0.999957	3.147960e-05	1.156381e-05	1.0	0.0	0.0	0.996672	0.001951	...	0.007440	0.995843	0.9
921	921	0	0.999393	6.059731e-04	1.219071e-06	0.0	1.0	0.0	0.996819	0.002231	...	0.005286	0.996066	0.9

980 rows × 23 columns

Function for entropy as anomaly measure

In [ ]: