In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers
import keras as keras
from keras import models
from keras import layers
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD
from keras.utils.np_utils import to_categorical
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.neural_network import MLPClassifier
from keras import metrics
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
Using TensorFlow backend.
```

In [2]:

```
red = pd.read_csv('data/winequality-red1.csv')
white = pd.read_csv('data/winequality-white1.csv')
```

In [3]:

```
red['quality'].replace(to_replace=[0,1,2,3,4,5], value=1, inplace=True)
red['quality'].replace(to_replace=[6], value=2, inplace=True)
red['quality'].replace(to_replace=[7,8,9,10], value=3, inplace=True)
X_red = red[
['fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol']]
Y_red = red[['quality']]
white['quality'].replace(to_replace=[0,1,2,3,4,5], value=1, inplace=True)
white['quality'].replace(to_replace=[6], value=2, inplace=True)
white['quality'].replace(to_replace=[7,8,9,10], value=3, inplace=True)
X_white = white[
['fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol']]

Y_white = white[['quality']]
X_red = X_red.values
Y_red = Y_red.values
X_white = X_white.values
Y_white = Y_white.values
number_of_features = 11
```

In [25]:

```
def rsme(targets, outputs):
    return tf.sqrt(tf.reduce_mean(tf.square(tf.subtract(targets, outputs))))

# Create function returning a compiled network
def create_network(activation):

    network = models.Sequential()
    if(activation == 'linear'):
        network.add(layers.Dense(units=64, activation='linear', input_shape=(number_of_feat
    elif(activation == 'sigmoid'):
        network.add(layers.Dense(units=64, activation='sigmoid', input_shape=(number_of_fea
    elif(activation == 'relu'):
        network.add(layers.Dense(units=64, activation='relu', input_shape=(number_of_featur
    elif(activation == 'tanh'):
        network.add(layers.Dense(units=64, activation='tanh', input_shape=(number_of_featur

    # Start neural network


    #network.add(layers.BatchNormalization())
    network.add(layers.Dense(3, activation='softmax'))

    #network.compile(loss='binary_crossentropy', # Cross-entropy
   #                 optimizer='rmsprop', # Root Mean Square Propagation
    #                metrics=['accuracy']) # Accuracy performance metric
    network.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy', rsme])

    # Return compiled network
    return network
```

In [26]:

```
neural_network = KerasClassifier(build_fn=create_network,
                                 epochs=10,
                                 batch_size=100,
                                 verbose=0)
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

In [18]:

```
print("sigmoid")
print("Red")
prediction_red = cross_val_predict(neural_network, X_red, Y_red, cv=cv)
print(classification_report(Y_red, prediction_red))
print("White")
prediction_white = cross_val_predict(neural_network, X_white, Y_white, cv=cv)
print(classification_report(Y_white, prediction_white))
```

```
Selu 64, BN, Softmax 3
Red
              precision    recall  f1-score   support

           1       0.48      0.50      0.49       744
           2       0.41      0.25      0.31       638
           3       0.14      0.27      0.18       217

   micro avg       0.37      0.37      0.37      1599
   macro avg       0.34      0.34      0.33      1599
weighted avg       0.40      0.37      0.38      1599

White
              precision    recall  f1-score   support

           1       0.35      0.37      0.36      1640
           2       0.45      0.65      0.53      2198
           3       0.30      0.00      0.01      1060

   micro avg       0.41      0.41      0.41      4898
   macro avg       0.37      0.34      0.30      4898
weighted avg       0.38      0.41      0.36      4898
```

In [20]:

```
print("Linear")
print("Red")
prediction_red = cross_val_predict(neural_network, X_red, Y_red, cv=cv)
print(classification_report(Y_red, prediction_red))
print("White")
prediction_white = cross_val_predict(neural_network, X_white, Y_white, cv=cv)
print(classification_report(Y_white, prediction_white))
```

```
Linear
Red
              precision    recall  f1-score   support

           1       0.49      0.21      0.30       744
           2       0.39      0.55      0.46       638
           3       0.15      0.26      0.19       217

   micro avg       0.35      0.35      0.35      1599
   macro avg       0.34      0.34      0.31      1599
weighted avg       0.40      0.35      0.35      1599

White
              precision    recall  f1-score   support

           1       0.32      0.26      0.29      1640
           2       0.45      0.66      0.54      2198
           3       0.16      0.05      0.08      1060

   micro avg       0.40      0.40      0.40      4898
   macro avg       0.31      0.33      0.30      4898
weighted avg       0.34      0.40      0.35      4898
```

In [23]:

```
print("tanh")
print("Red")
prediction_red = cross_val_predict(neural_network, X_red, Y_red, cv=cv)
print(classification_report(Y_red, prediction_red))
print("White")
prediction_white = cross_val_predict(neural_network, X_white, Y_white, cv=cv)
print(classification_report(Y_white, prediction_white))
```

```
tanh
Red

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-23-378e09250550> in <module>()
      1 print("tanh")
      2 print("Red")
----> 3 prediction_red = cross_val_predict(neural_network, X_red, Y_red, cv=
cv)
      4 print(classification_report(Y_red, prediction_red))
      5 print("White")

~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\model_selectio
n\_validation.py in cross_val_predict(estimator, X, y, groups, cv, n_jobs, v
erbose, fit_params, pre_dispatch, method)
    775     prediction_blocks = parallel(delayed(_fit_and_predict)(
    776         clone(estimator), X, y, train, test, verbose, fit_params, me
thod)
--> 777         for train, test in cv.split(X, y, groups))
    778 
    779     # Concatenate the predictions

~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\jobl
ib\parallel.py in __call__(self, iterable)
    984                 self._iterating = self._original_iterator is not Non
e
    985 
--> 986             while self.dispatch_one_batch(iterator):
    987                 pass
    988 

~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\jobl
ib\parallel.py in dispatch_one_batch(self, iterator)
    823                 return False
    824             else:
--> 825                 self._dispatch(tasks)
    826                 return True
    827 

~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\jobl
ib\parallel.py in _dispatch(self, batch)
    780         with self._lock:
    781             job_idx = len(self._jobs)
--> 782             job = self._backend.apply_async(batch, callback=cb)
    783             # A job can complete so quickly than its callback is
    784             # called before we get here, causing self._jobs to
```

```
~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\jobl
ib\_parallel_backends.py in apply_async(self, func, callback)
    180     def apply_async(self, func, callback=None):
    181         """Schedule a func to be run"""
--> 182         result = ImmediateResult(func)
    183         if callback:
    184             callback(result)

~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\jobl
ib\_parallel_backends.py in __init__(self, batch)
    543         # Don't delay the application, to avoid keeping the input
    544         # arguments in memory
--> 545         self.results = batch()
    546 
    547     def get(self):

~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\jobl
ib\parallel.py in __call__(self)
    259         with parallel_backend(self._backend):
    260             return [func(*args, **kwargs)
--> 261                     for func, args, kwargs in self.items]
    262 
    263     def __len__(self):

~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\jobl
ib\parallel.py in <listcomp>(.0)
    259         with parallel_backend(self._backend):
    260             return [func(*args, **kwargs)
--> 261                     for func, args, kwargs in self.items]
    262 
    263     def __len__(self):

~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\model_selectio
n\_validation.py in _fit_and_predict(estimator, X, y, train, test, verbose,
 fit_params, method)
    850         estimator.fit(X_train, y_train, **fit_params)
    851     func = getattr(estimator, method)
--> 852     predictions = func(X_test)
    853     if method in ['decision_function', 'predict_proba', 'predict_log
_proba']:
    854         n_classes = len(set(y))

~\AppData\Local\Continuum\anaconda3\lib\site-packages\keras\wrappers\scikit_
learn.py in predict(self, x, **kwargs)
    232         else:
    233             classes = (proba > 0.5).astype('int32')
--> 234         return self.classes_[classes]
    235 
    236     def predict_proba(self, x, **kwargs):

IndexError: index 21 is out of bounds for axis 1 with size 3
```

In [ ]:

```

```