Ensemble Learning Report

Xining Li, Debashis Kayal

1 Motivation

Machine learning has already become ubiquitous in all aspects of modern-day living. Predictions from machine learning models are increasingly becoming an integral part of most computational systems around us. With the increased usage, achieving a higher prediction accuracy is an obvious need of the hour.

In particular, existing machine learning algorithms are trying to find function within the hypothesis space that can approximate the following formulas.

$$f^*(x) = \text{argmax}_{Y=y} P(Y = y|X = x) \tag{1}$$

The above formula does not address the existence of mislabeled data. Therefore, any classier's performance will be harmed by noise in dataset labels, which is to be anticipated; the relevant question is by how much?

Since machine learning's prediction is directly dependent on the dataset that the models are trained upon, it follows that a high quality - properly labeled, de-duped, correct data is required.

However in real life high quality data is hard to obtain - with the ever increasing "noises" which may cause data error. Typical data faults include:

- Repetitions (duplication)
- Mislabelings (label noise)
- Distortions (intentional or unintentional deviation of data from its true or most accurate representation of the full picture)

2 Label noise in ML

Dataset noise can be either feature noise or label noise. Considering the relatively very high impact of label noise, we will focus on understanding its impacts. Researchers Some research [1] has shown that ML systems can generalize and behave well even when trained with massive mislabeled data.

In particular, ML systems are robust with label noise dependent on geospatial proximity, or geospatial label noise [2].

However the ML systems' performance may be penalized under certain specific circumstances:

- The ML system must have a large enough parameter set to manage the complexity of the image classification task.
- The training dataset must be very large – large enough to include many properly labeled examples, even if most of the training data is mislabeled. With enough good training samples, the ML system can learn accurate classification by finding the 'signal' buried in the label noise.
- The mislabeling is random and does not form a pattern of its own [3]

However most ML models face challenges that make label noise a sufficiently big problem. These pose complications such as:

- Unavailability of large training datasets
- Systematic, pattern-building labeling errors that confuse machine learning.

The performance of any classifier, or for that matter any machine learning task, depends crucially on the quality of the available data.

Aim of this paper is to explore the usage of ensemble methods in machine learning and how it can help increase the fault-tolerancy of the models.

It has already been established that Ensemble Machine Learning improves accuracy by reducing the variance compared to using a single model. Additionally, ensemble techniques also improve the average performance of a model thus contributing to the robustness of the model. Some ensemble methods like boosting (XGb) are robust enough to take care of missing data on their own.

3 Ensemble Learning in ML

The premise of using ensemble-based decision systems in our daily lives is fundamentally not different from their use in computational intelligence. We consult with others before making a decision often because of the variability in the past record and accuracy of any of the individual decision-makers [4].

Ensemble method is the machine learning technique that involves combining several base models to create a better predictive model [5] [6]. Idea is that accuracy of the ensemble model is much better than the individual learner models. By using ensemble methods we aim to gain:

- Higher predictive accuracy (lower errors)
- More consistency (by avoiding overfitting)
- Reduce bias and variance errors

4 Different Popular Techniques of Ensemble Learning

Popular ensemble learning models include

- Random Forest: Selecting subset of traning data and features and build decision trees.
- Bagging: Random sample of data in a training set is selected with replacement
- Boosting: Building models in sequence, and minimizing the errors by combining a set of weak learners into a strong learners [7].
- Gradient Boosting: generating a prediction model from an ensemble of weak models in sequence (typically decision trees)
- XGBoost: regularizing gradient boosting [8] [9]

4.1 Bagging: Bootstrap Aggregating

Using the same learning models trained with different slices (subsets) of data randomly sliced from the whole training dataset and then voting (majority) to arrive at the output.

1. Random splitting of the training data set into multiple samples. There may be some data overlap between samples (since data slicing is random).
2. Base model selected is trained using the bootstrap samples The selection of samples is also random thus the same sample may be used to train more than one model.
3. Aggregating - the output from the different base models is used to arrive at the final output.

For classification problems, a voting mechanism (usually majority vote) is used.

For regression problems, the average of the output from the individual models is taken to arrive at the ensemble system output.

Both averaging and voting can be simple or weighted if relevant weights are available for use.

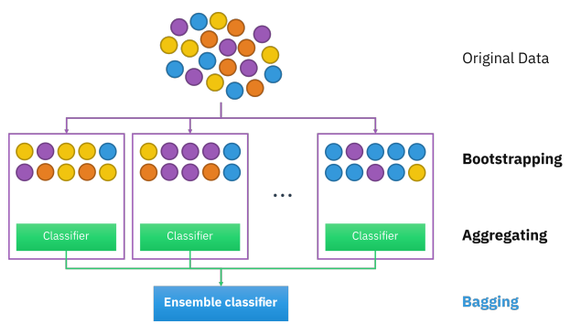

Figure 1: Bagging [10]

In Bagging, base models can train in parallel [11]. Bagging method is aimed at reducing variance. Since individual models can train in parallel - bagging is a parallel method of ensemble techniques.

Random Forests is a bagging method where deep trees are trained on bootstrap samples and combined to produce output with lower variance.In addition to the dataset sampling, it does sampling over features and uses a random subset of those features to build the tree. This ensures that not all decision trees take in the exact same training data. This sampling over features makes the process more robust to missing data as the trees take into account only features where data are not missing.

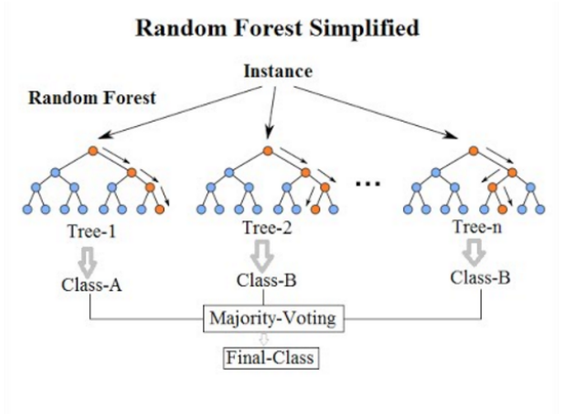

Figure 2: Random forest is a bagging method that combines bagging with random feature subset selection [12]

4.2 Boosting

Similar to bagging uses several base models but in a sequential iterative manner. Each sequentially fitted base learner focuses on improving the error output from the previous model. Thus we get a method that has lower bias and reduced variance. Boosting is mainly focused at reducing bias. Two types of boosting techniques - Adaptive Boosting (ada boost) and Gradient boosting. In Adaboost - the weight attached to each observation is changed while in Gradient Boosting the actual observation values are changed

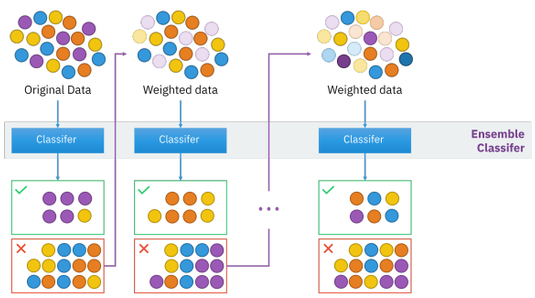

Figure 3: Adaptive Boosting or Adaboost [13]

Gradient boosting does not modify the sample distribution as weak learners train on the remaining residual errors (i.e, pseudo-residuals) of a previous learner. By training on the residuals of the model, is an alternative means to give more importance to misclassified observations.

Xtreme gradient boosting or XGB is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. XGBoost is basically designed to enhance the performance and speed of a Machine Learning model. In prediction problems involving unstructured data (images, text, etc.), artificial neural networks tend to outperform all other algorithms or frameworks [14]. Like gradient boosting XGBoost also handles missing values in the dataset.

A generic unregularized xgboost algorithm is described clearly in [8] and [15].

5 Objectives of this Paper

In this project, we will be investigating the performance of ensemble learning algorithms with mislabeled data. We will be exploring dierent datasets, dierent machine learning algorithms, and dierent machine learning approaches; we will be juxtaposing training data without mislabeling and with mislabeling under the combinations of these. For the scope of this paper we have identified two major field of applications of ML: image recognition and natural language processing (NLP). Dataset Sourcing: We will be sourcing our datasets from publicly available standard datasets (MNIST, CIFAR10. . .) for Image Recognition.

6 Experiment

6.1 MINST Dataset [16]

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples.

```
dirname = os.path.dirname(__file__)
os.makedirs(os.path.join(dirname, "../report-resources/minst"), exist_ok=True)
train = pd.read_csv(os.path.join(dirname, "../all-data/minst/train.csv"))
y_train, x_train = train.label.values, train.drop("label", axis=1).values
plt.imshow(x_train[0].reshape(28, 28), interpolation="gaussian")
```

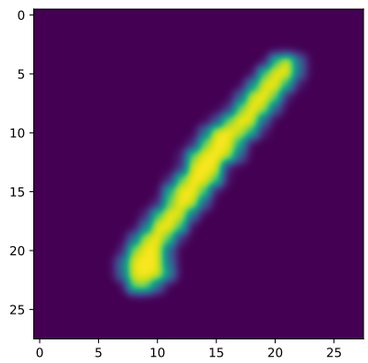

Figure 4: MINST data handwriting example

The Figure 4 is a letter 1 in the MINST dataset.

6.1.1 Without Data Mislabeling

The following algorithms run on the data without mislabeling.

```
lg_clf = LogisticRegression(solver='lbfgs', max_iter=1000)
lg_clf.fit(x_train, y_train)
logistic_regression_validate_predict = lg_clf.predict(x_validate)
logistic_regression_f1_score = f1_score(
    y_validate,
    logistic_regression_validate_predict,
    average='weighted')
print(logistic_regression_f1_score)

decision_tree_clf = tree.DecisionTreeClassifier()
decision_tree_clf.fit(x_train, y_train)
decision_tree_validate_predict = decision_tree_clf.predict(x_validate)
decision_tree_f1_score = f1_score(
    y_validate,
    decision_tree_validate_predict,
    average='weighted')
print(decision_tree_f1_score)

random_forest_clf = RandomForestClassifier(n_jobs=-1)
random_forest_clf.fit(x_train, y_train)
random_forest_validate_predict = random_forest_clf.predict(x_validate)
random_forest_f1_score = f1_score(
    y_validate,
    random_forest_validate_predict,
    average='weighted')
print(random_forest_f1_score)

from sklearn.ensemble import AdaBoostClassifier
ada_clf = AdaBoostClassifier(random_state=0)
ada_clf.fit(x_train, y_train)
ada_validate_predict = ada_clf.predict(x_validate)
ada_f1_score = f1_score(y_validate, ada_validate_predict, average='weighted')
print(ada_f1_score)

import xgboost as xgb
xgb_clf = xgb.XGBRegressor(objective="multi:softmax", num_class=10, random_state=42)
xgb_clf.fit(x_train, y_train)
xgb_validate_predict = xgb_clf.predict(x_validate)
xgb_f1_score = f1_score(y_validate, xgb_validate_predict, average='weighted')
print(xgb_f1_score)

simple_comparison_result = pd.DataFrame(dict(
    logistic_regression=[logistic_regression_f1_score],
    decision_tree=[decision_tree_f1_score],
    random_forest=[random_forest_f1_score],
    ada = [ada_f1_score],
    xgb = [xgb_f1_score]
    ))
```

logistic_regression	decision_tree	random_forest	ada	xgb
0.899142	0.856739	0.964156	0.718159	0.972569

Figure 5: F1 Scores Table

6.1.2 With Data Mislabeling

We use the following data to intentionally create misslabeled data

```
import numpy as np
import random
def do_miss_label(y_train_labels, miss_labeled_ratio):
    y_train_copy = np.copy(y_train_labels)
    all_labels = set(y_train_copy)
    all_labels_count = len(all_labels)
    misslabeled_indices = np.random.choice(len(y_train_copy)-1,
                                           int((len(y_train_copy)-1)*miss_labeled_ratio))
    for misslabeled_index in misslabeled_indices:
        y_train_copy[misslabeled_index] = random.randrange(0,all_labels_count)
    return y_train_copy
```

```
lg_clf_misslabeled = LogisticRegression(solver='lbfgs', max_iter=1000)
lg_clf_misslabeled.fit(x_train, y_train_miss_label)
logistic_regression_misslabeled_validate_predict = lg_clf_misslabeled.predict(x_validate)
logistic_regression_f1_score_misslabeled = f1_score(y_validate,
                                        logistic_regression_misslabeled_validate_predict,
                                        average='weighted')
print(logistic_regression_f1_score_misslabeled)

decision_tree_clf_misslabeled = tree.DecisionTreeClassifier()
decision_tree_clf_misslabeled.fit(x_train, y_train_miss_label)
decision_tree_misslabeled_validate_predict = decision_tree_clf_misslabeled.predict(x_validate)
decision_tree_f1_score_misslabeled = f1_score(y_validate,
                                  decision_tree_misslabeled_validate_predict,
                                  average='weighted')
print(decision_tree_f1_score_misslabeled)

random_forest_clf_misslabeled = RandomForestClassifier(n_jobs=-1)
random_forest_clf_misslabeled.fit(x_train, y_train_miss_label)
random_forest_misslabeled_validate_predict = random_forest_clf_misslabeled.predict(x_validate)
random_forest_misslabeled_f1_score = f1_score(y_validate,
                                              random_forest_misslabeled_validate_predict,
                                              average='weighted')
print(random_forest_misslabeled_f1_score)

import xgboost as xgb
xgb_clf_misslabeled = xgb.XGBRegressor(objective="multi:softmax",
    num_class=10, random_state=42)
xgb_clf_misslabeled.fit(x_train, y_train_miss_label)
xgb_validate_predict_misslabeled = xgb_clf_misslabeled.predict(x_validate)
xgb_f1_score_misslabeled = f1_score(y_validate,
    xgb_validate_predict_misslabeled,
    average='weighted')
print(xgb_f1_score_misslabeled)

from sklearn.ensemble import AdaBoostClassifier
ada_clf_misslabeled = AdaBoostClassifier(random_state=0)
ada_clf_misslabeled.fit(x_train, y_train_miss_label)
ada_validate_predict_misslabeled = ada_clf_misslabeled.predict(x_validate)
ada_f1_score_misslabeled = f1_score(y_validate, ada_validate_predict_misslabeled, average='weighted')
print(ada_f1_score_misslabeled)


misslabeled_comparison_result = pd.DataFrame(dict(
    logistic_regression=[logistic_regression_f1_score_misslabeled],
    decision_tree=[decision_tree_f1_score_misslabeled],
    random_forest=[random_forest_misslabeled_f1_score],
    ada = [ada_f1_score_misslabeled],
    xgb = [xgb_f1_score_misslabeled]
    ))
```

logistic_regression	decision_tree	random_forest	ada	xgb
0.863002	0.620582	0.96224	0.78974	0.948439

Figure 6: Misslabled data F1 Scores Table

6.1.3 Aggregated View

Models		Without Mislabelled Data	With Mislabelled Data
Base Model	logistic_regression	0.899142	0.863002
Base Model	decision_tree	0.856739	0.620582
Ensemble	Random_forest	0.964156	0.96224
Ensemble	Ada Boost	0.718159	0.78974
Ensemble	XGBoost	0.972569	0.948439

6.2 The CIFAR-10 dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:

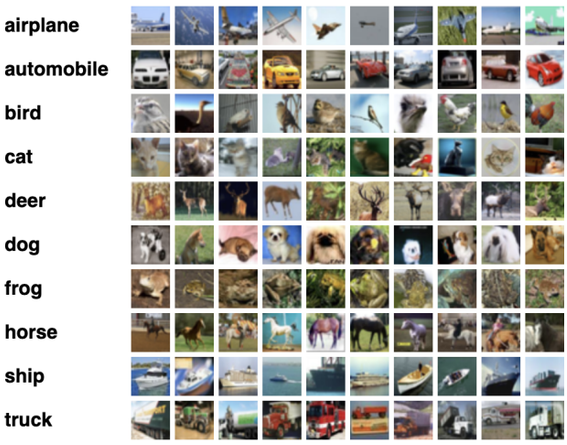

Figure 7: CFAR10 data example [17]

CFAR10 is a more complicated dataset, and we will be using the CNN for the feature extraction. After the feature extraction, we will be using the following ways for the classification.

- Softmax
- Random Forest

6.2.1 Without Data Mislabeling

```
import time
# import keras
from keras.datasets import cifar10
from tensorflow.keras.models import Sequential


from tensorflow.keras import datasets, layers, models
from keras.utils import np_utils
from tensorflow.keras import regularizers
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
import matplotlib.pyplot as plt
import numpy as np

# example of loading the cifar10 dataset
from matplotlib import pyplot
from keras.datasets import cifar10
# load dataset
(train_images, train_labels), (test_images, test_labels)= cifar10.load_data()
# summarize loaded dataset
print('Train: X=%s, y=%s' % (train_images.shape, train_labels.shape))
print('Test: X=%s, y=%s' % (test_images.shape, test_labels.shape))
# plot first few images
for i in range(9):
        # define subplot
        pyplot.subplot(330 + 1 + i)
        # plot raw pixel data
        pyplot.imshow(train_images[i])
# show the figure
pyplot.show()


# one hot encode target values
from keras.utils import to_categorical

# Converting the pixels data to float type
train_images = train_images.astype('float32')
test_images = test_images.astype('float32')

# Standardizing (255 is the total number of pixels an image can have)
train_images = train_images / 255
test_images = test_images / 255

# One hot encoding the target class (labels)
num_classes = 10
train_labels = to_categorical(train_labels, num_classes)
```

```
test_labels = to_categorical(test_labels, num_classes)
# Creating a sequential model and adding layers to it
feature_extractor = Sequential()
feature_extractor.add(layers.Conv2D(32, (3,3), padding='same', activation='relu',
    input_shape=(32,32,3)))
feature_extractor.add(layers.BatchNormalization())

feature_extractor.add(layers.Conv2D(32, (3,3), padding='same', activation='relu'))
feature_extractor.add(layers.BatchNormalization())
feature_extractor.add(layers.MaxPooling2D(pool_size=(2,2)))
feature_extractor.add(layers.Dropout(0.3))

feature_extractor.add(layers.Conv2D(64, (3,3), padding='same', activation='relu'))
feature_extractor.add(layers.BatchNormalization())

feature_extractor.add(layers.Conv2D(64, (3,3), padding='same', activation='relu'))
feature_extractor.add(layers.BatchNormalization())
feature_extractor.add(layers.MaxPooling2D(pool_size=(2,2)))
feature_extractor.add(layers.Dropout(0.5))

feature_extractor.add(layers.Conv2D(128, (3,3), padding='same', activation='relu'))
feature_extractor.add(layers.BatchNormalization())

feature_extractor.add(layers.Conv2D(128, (3,3), padding='same', activation='relu'))
feature_extractor.add(layers.BatchNormalization())
feature_extractor.add(layers.MaxPooling2D(pool_size=(2,2)))
feature_extractor.add(layers.Dropout(0.5))

feature_extractor.add(layers.Flatten())
feature_extractor.add(layers.Dense(128, activation='relu'))
feature_extractor.add(layers.BatchNormalization())
feature_extractor.add(layers.Dropout(0.5))


from tensorflow.keras.models import Model
x = feature_extractor.output
prediction_layer = Dense(num_classes, activation = 'softmax')(x)
cnn_model = Model(inputs=feature_extractor.input, outputs=prediction_layer)

from tensorflow.keras.optimizers import Adam, SGD
learning_rate = 0.001
cnn_model.compile(loss="categorical_crossentropy",
    optimizer=Adam(learning_rate=learning_rate), metrics=['accuracy'])

history = cnn_model.fit(train_images, train_labels, batch_size=64, epochs=10,
                    validation_split=0.1)


pred = cnn_model.predict(test_images)
print(pred)
# Converting the predictions into label index
pred_classes = np.argmax(pred, axis=1)
print(pred_classes)

from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
cnn_f1_score = f1_score(np.argmax(test_labels, axis=1), np.array(pred_classes), average='weighted')
cnn_accuracy = accuracy_score(np.argmax(test_labels, axis=1), np.array(pred_classes))

print("cnn_f1_score: " + str(cnn_f1_score))
print("cnn_accuracy: " + str(cnn_accuracy))


from keras.models import Model
X_for_RF = feature_extractor.predict(train_images) #This is out X input to RF
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_jobs=-1)
rf.fit(X_for_RF, np.argmax(train_labels, axis=1)) #For sklearn no one hot encoding
X_test_feature = feature_extractor.predict(test_images)
prediction_RF = rf.predict(X_test_feature)
from sklearn.metrics import f1_score
cnn_rf_f1=f1_score(np.argmax(test_labels, axis=1), prediction_RF, average='weighted')
cnn_rf_accuracy=accuracy_score(np.argmax(test_labels, axis=1), prediction_RF)

print("cnn_rf_f1_score: " + str(cnn_rf_f1))
print("cnn_rf_accuracy: " + str(cnn_rf_accuracy))
```

CNN	CNN+RandomForest
0.7885768705224767	0.8059692064910149

6.2.2 With Data Mislabeling

We applied 30% of mislabeled data.

```
original_train_labels = train_labels
import numpy as np
```

```
import random
def do_miss_label(y_train_labels, miss_labeled_ratio):
    y_train_copy = np.copy(y_train_labels)
    all_labels = set(y_train_copy)
    all_labels_count = len(all_labels)
    misslabeled_indices = np.random.choice(len(y_train_copy)-1,
                                           int((len(y_train_copy)-1)*miss_labeled_ratio))
    for misslabeled_index in misslabeled_indices:
        y_train_copy[misslabeled_index] = random.randrange(0,all_labels_count)
    return y_train_copy

train_labels = np.array(
    [
        do_miss_label(
            list(
            map(lambda e: e[0], original_train_labels)
            ), 0.3
            )
        ]
    ).T
```

CNN	CNN+RandomForest
0.7371853465743461	0.7546625150853542

6.2.3 Aggregated View

Models		Without Mislabelled Data	With Mislabelled Data
Base Model	CNN	0.7885768705224767	0.7371853465743461
Ensemble	CNN+ Random_forest	0.8059692064910149	0.7546625150853542

References

[1] D. Rolnick, A. Veit, S. Belongie, and N. Shavit, "Deep learning is robust to massive label noise," Feb 2018. [Online]. Available: https://arxiv.org/abs/1705.10694

[2] J. Frank, U. Rebbapragada, J. Bialas, T. Oommen, and T. C. Havens, "Effect of label noise on the machine-learned classification of earthquake damage," Apr 2017. [Online]. Available: https://digitalcommons.mtu.edu/michigantech-p/924

[3] i. , "How noisy labels impact machine learning models," Apr 2021. [Online]. Available: https://www.kdnuggets.com/2021/04/imerit-noisy-labels-impact-machine-learning.html

[4] G. D. Martin, "Mnist with ensemble methods," Mar 2018. [Online]. Available: https://www.kaggle.com/grrddm/mnist-with-ensemble-methods

[5] W. , "Ensemble learning," Sep 2021. [Online]. Available: https://en.wikipedia.org/wiki/Ensemble_learning

[6] C. Zhang and Y. Ma, "Ensemble machine learning: Methods and applications," 2012.

[7] "Boosting (machine learning)," Jul 2021. [Online]. Available: https://en.wikipedia.org/wiki/Boosting_(machine_learning)#/media/File:Ensemble_Boosting.svg

[8] Wikipedia contributors, "Xgboost — Wikipedia, the free encyclopedia," 2021, [Online; accessed 8-September-2021]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=XGBoost&oldid=1035329027

[9] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," Jun 2016. [Online]. Available: https://arxiv.org/abs/1603.02754

[10] W. Commons, "File:ensemble bagging.svg — wikimedia commons, the free media repository," 2020, [Online; accessed 8-September-2021]. [Online]. Available: https://commons.wikimedia.org/w/index.php?title=File:Ensemble_Bagging.svg&oldid=504660847

[11] J. Rocca, "Ensemble methods: Bagging, boosting and stacking," Mar 2021. [Online]. Available: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

[12] W. Commons, "File:random forest diagram complete.png — wikimedia commons, the free media repository," 2021, [Online; accessed 8-September-2021]. [Online]. Available: https://commons.wikimedia.org/w/index.php?title=File:Random_forest_diagram_complete.png&oldid=558012224

[13] ——, "File:ensemble boosting.svg — wikimedia commons, the free media repository," 2020, [Online; accessed 8-September-2021]. [Online]. Available: https://commons.wikimedia.org/w/index.php?title=File:Ensemble_Boosting.svg&oldid=505336553

[14] H. Mujtaba, "What is gradient boosting and how is it different from adaboost?" Mar 2021. [Online]. Available: https://www.mygreatlearning.com/blog/gradient-boosting/

[15] V. Morde, "Xgboost algorithm: Long may she reign!" Apr 2019. [Online]. Available: https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

[16] Wikipedia contributors, "Mnist database — Wikipedia, the free encyclopedia," 2021, [Online; accessed 8-September-2021]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=MNIST_database&oldid=1039756601

[17] ——, "Cifar-10 — Wikipedia, the free encyclopedia," 2021, [Online; accessed 8-September-2021]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=CIFAR-10&oldid=1014793182