DMS140_CW1_Menna_MAttia_SN_210102767

December 19, 2021

0.1 1. Introduction

In the modern era, we are blessed with an enormous quantity of information. Everything can be found online, probably every bit of human knowledge is available with a few clicks, and most of it is completely free. The downside of the connected world is probably the fact that there's no filter, everyone with a smartphone can post information online, and the more this information is repeated and liked (or disliked) in general the more it will be potent and influential (and also profitable), regardless of its veridicality. With the term fake news, we identify any false or misleading information presented as news [1], and we have seen them interfere with elections, COVID-19 vaccination programs, and ruin the reputation of many people in the last few years. The problem of automatically detecting fake news it's not an easy one to solve, in this paper we will briefly look at the state of the art and try to add novelty to a particular approach.

0.1.1 1.1 Domain-specific Area

According to [2] the problem of automatically detecting fake news is not easy to solve because of two factors:

1. The Fake news content can be images or video or a podcast, very easy to fake but a lot more complex to analyze and preprocess than normal text.
2. There is no way in knowing where people take their information from. The web is full of platforms that provide news and governance is basically non-existent

But nonetheless, the ML community has developed a series of solutions to tackle the problem with promising results (at least in the text-based news domain). In the survey by Shivam B. Parikh and Pradeep K. Atrey [3] the approaches are divided into six methodology groups:

1. Linguistic Features-based Methods, based on the extraction and classification of linguistic features from fake news, usually using a tf-idf representation of the text.
2. Deception Modeling based Methods, based on the extraction of the relations between text units on a story as a hierarchical tree.
3. Clustering based Methods, based on agglomerative clustering algorithms (such as KNN) trained on a large number of data sets.
4. Predictive Modeling based Methods, based on logistic regression and positive or negative coefficients to point out the deception probability of a given text.
5. Content Cues based Methods, based on the assumption that the fake news is created solely to engage the readers, unlike real news, and some form of the linguistic pattern are an indicator of this purpose
6. Non-Text Cues based Methods, focuses on the analysis of two non-text components of news: images and user behavior.

In this work the focus will be on methodologies of class 1, the text will be processed and stored as a tf-idf matrix and different models will be evaluated against a baseline.

0.1.2 1.2 Description of the selected dataset

The datasets used for this analysis are three:

The LIAR dataset presented in [4] and available to the public. It is composed of 12,8 K human labeled short statements from politifact.com labeled with truthfulness ratings: pants-fire, false, barely-true, half-true, mostly-true, and true. The dataset is well balanced and since the analysis will be a binary one (fake news yes/no), to mantain balance we apply the following mapping:

- pants-fire: fake news
- false: fake news
- barely-true: fake news
- half-true: true
- mostly-trye: true
- true: true

The LIAR dataset is downloaded as three tsv files divided into train, test, and validation sets. It is composed of the following columns:

1. ID - Text
2. Label - Text
3. Statement - Text
4. Subject - Text
5. Speaker - Text
6. Speaker Job Title - Text
7. State - Text
8. Party affiliation - Text - [democrat, republican]
9. Barely true count - Integer
10. Half true counts - Integer
11. Mostly true counts - Integer
12. Pants on fire counts - Integer
13. Venue/location of the statement - Text

The second dataset is the ISOT Fake News Dataset, introduced by Ahmed H, Traore I and Saad S. in [5], [6] and available on Kaggle [7]. It is composed of 21417 true news articles and 23481 fake news. The truthful articles were obtained by crawling articles from Reuters.com, and the fake news from different sources, mostly unreliable websites flagged by Politifact and Wikipedia.

The ISOT dataset is downloaded as two csv files, true.csv, and fake.csv. It is composed of the following columns:

1. Title - Text
2. Text - Text
3. Subject - Text
4. Date - Date

Both the described datasets will be reduced to the same format for this analysis:

1. Article - Text
2. isFake - Boolean

From the LIAR dataset we'll sample 3K random rows from the train file and from the ISOT dataset we'll sample 1,5K random rows from the true file and 1,5k rows from the fake file.

The third dataset used is a validation dataset and is the concatenation of the previous two datasets. It will have the standard format and it'll be composed of 6K rows.

0.1.3 1.3 Objectives

The objectives of this project are mainly two:

1. Explore different ensemble methodologies with a set of classificators that will be the baseline against which the ensembles will be evaluated. Study the differences between those methodologies and find if there's one best suited to the task of finding fake news. The ensemble techniques that will be used are:

   Hard Blending Ensemble, a form of Stacking Generalization without the k-fold cross-validation. We use the predictions of the base models to create a "meta-model" that will be then used as training for a "blending model" (in our case Logistic Regressor) that will do the actual predictions.

   Soft Blending Ensemble, like above, but with the difference that instead of using the predictions of the base models as meta-model we will use the probabilities given by the models as training for the blender

   Soft Weighted Voting Ensemble, a form of Voting Ensemble in which the predictions of the base models will result in a prediction based on the majority vote, with a weight given by the accuracy of the single base model on a validation set

2. Create an ensemble that can outperform any of the single "weak learners" composing the ensemble classificator.

0.1.4 1.4 Evaluation Methodology

For the evaluation of the algorithms three confusion matrix based scores will be used:

Precision

Precision is the rate of positive instances that were correctly identified by the classificator over all the positive predicted instances, it is calculated as: $\frac{TP}{TP+FP}$, where TP is the True Positive and

FP is the False Positive. The precision is a measure of the accuracy of the classificator over the positive examples. In the case of a binary classificator a precision of 1 means that all the positive examples predicted by the model are really positive.

Recall

Recall is the rate of positive instances that were correctly identified by the classificator over all the real positive instances, it is calculated as $\frac{TP}{TP+FN}$, where FN is the False Negative. In the case of a binary classificator a recall of 1 means that all the really positive examples have been correctly predicted by the model.

Accuracy

Accuracy is the rate of correct prediction identified by the classificator over the entire sample, it is often expressed as the percentage of correct answers given against a test set. It is calculated as $\frac{TP+TN}{TP+TN+FP+FN}$

F1-Score

F1 Score is the harmonic mean of the precision and recall, it is calculated as $2 * \frac{Precision*Recall}{Precision+Recall}$. It takes into consideration both precision and recall and gives an overall score of the model. An F-score of 1 means that the model has perfect precision and recall, an f-score of 0 means that either recall or precision is 0.

The evaluation with these metrics will be carried out on a training and test set, both for the baseline and the ensembles. The values of all three metrics are higher for higher performance algorithms and have a range 0-1. The training and test sets will be split as 80% training and 20% test.

0.2 2. Implementation

[1]:
```python
#Utilities
import pandas as pd
import nltk
import string
import random
import numpy as np
from numpy import hstack
from nltk.stem.snowball import SnowballStemmer
from nltk import ngrams
from functools import reduce
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.utils.extmath import softmax

nltk.download('stopwords')

#TF-IDF
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

#Models
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import RidgeClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn import tree
from sklearn.svm import SVC

#Evaluation Metrics
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/mmenna/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

0.2.1 2.1 Pre-processing

The preprocessing of the text will consist of the tokenization of the text and the codification of the entire corpus in a tf-idf matrix.

The tokenization of the text is implemented in the custom function "text-processing" below, it performs 5 steps:

1. Convert text to lower case, to avoid differentiation between words with or without a capital letter
2. Remove all punctuation, that is considered not relevant to the analysis
3. Remove stopwords, to remove the uninformative pieces of text and reduce the size of the matrix
4. Apply stemming with the Snowball Stemmer, to avoid differentiation between different verb forms, plural and singular, etc… for the same word, the word is reduced to his "stem" (es. going, go= go)
5. Tokenise the text with n grams, to split the text in ngrams

Then the entire corpus of tokenized text is converted in a tf-idf matrix with the tfIdfCustomTransformer function below, which leverages on the sklearn libraries. The tf-idf matrix is a numerical matrix that reflects how important a word is to a document in a corpus, every document becomes a vector that is then used as feature to train the models.

[2]:
```
def text_processing(text, n = 1):
    """
    Takes in a string of text, then performs the following:
    1. Convert text to lower case and remove all punctuation
    2. Remove stopwords
    3. Apply stemming
    4. Apply Ngram Tokenisation
```

```
    5. Returns the tokenised text as a list

    Parameters
    ----------
    text : string
        String to pre-process
    n : int
        Number of ngrams for the tokenization

    Returns
    -------
    tokenised : list
        List containing the tokenised text
    """

    stemmer = SnowballStemmer("english")
    stop = stopwords.words('english')
    #write steps here
    # lower function
    t_1 = lambda x : x.lower()
    # Remove punctuation function
    t_2 = lambda x : x.translate(str.maketrans('', '', string.punctuation))
    # Remove stopwords
    t_3 = lambda x : " ".join([w for w in x.split() if w not in stop])
    # Snowball stemming
    t_4 = lambda x : " ".join([stemmer.stem(w) for w in x.split()])
    # Ngrams with n number of grams
    t_5 = lambda x : [" ".join(ng) for ng in list(ngrams(x.split(), n))]


    #List of transformation functions
    t = [t_1, t_2, t_3, t_4, t_5]

    #Apply transformations
    tokenised = reduce(lambda r, f: f(r), t, text)

    return tokenised
```

```
[3]: def tfIdfCustomTransformer(corpus, preprocessor):
    """
    Takes in an of string, then performs the transformation in a tf-idf matrix␣
 ↪with
    custom pre-processor and tokenizer

    Parameters
    ----------
    corpus : list of string
```

```
        List of documents to transform to tdf-idf matrix
    preprocessor : function
        Preprocessing function to apply to the corpus prior
        the tf-idf transformation

    Returns
    -------
    text_tfidf : matrix
        TF-IDF matrix
    """
    identity = lambda x : x
    vectorizer = CountVectorizer(tokenizer = identity, preprocessor = identity)
    tfidfTransformer = TfidfTransformer()

    bag = [preprocessor(x) for x in corpus]
    count_vector = vectorizer.fit_transform(corpus).toarray()
    text_tfidf = tfidfTransformer.fit_transform(count_vector)

    return text_tfidf.toarray()
```

[4]:

```
#Reding ISOT dataset from CSV
isot_raw_data_true = pd.read_csv('True_sample.csv')
isot_raw_data_fake = pd.read_csv('Fake_sample.csv')
#Creating the isFake column and appending the two files in a single dataframe
isot_raw_data_true['isFake'] = 0
isot_raw_data_fake['isFake'] = 1
isot_raw_data = isot_raw_data_true.append(isot_raw_data_fake)
#Reducing the ISOT dataset as a article-isfake format in the i_data_t1 dataframe
i_data_t1 = pd.DataFrame()
i_data_t1['article'] = isot_raw_data['title'] + ' ' + isot_raw_data['text']
i_data_t1['isFake'] = isot_raw_data['isFake']

#Reading the LIAR dataset from csv
liar_raw_data = pd.read_csv('LIAR_sample.csv')
#Mapping the labels to obtain a binary label
liar_mapper = {
    'false': 1,
    'half-true': 0,
    'mostly-true': 0,
    'true': 0,
    'barely-true': 1,
    'pants-fire': 1
}
reduce_fake = lambda x : liar_mapper[x]
l_data_t1 = pd.DataFrame()
#Reducing the LIAR dataset as a article-isfake format in the l_data_t1 dataframe
l_data_t1['article'] = liar_raw_data['Statement']
```

```
l_data_t1['isFake'] = liar_raw_data['Label'].apply(reduce_fake)

#Creating the TOTAL dataset as concatenation of LIAR and ISOT in standard format
t_data_t1 = i_data_t1.append(l_data_t1)
```

[5]:
```
# ISOT dataset features and labels
i_X = pd.DataFrame(tfIdfCustomTransformer(i_data_t1['article'].values,␣
 ↪text_processing))
i_y = i_data_t1['isFake']

# LIAR dataset features and labels
l_X = pd.DataFrame(tfIdfCustomTransformer(l_data_t1['article'].values,␣
 ↪text_processing))
l_y = l_data_t1['isFake']

# TOTAL dataset features and labels
t_X = pd.DataFrame(tfIdfCustomTransformer(t_data_t1['article'].values,␣
 ↪text_processing))
t_y = t_data_t1['isFake']
```

0.2.2 2.2 Baseline performance

The baseline performance for this work is the performance of 6 of the most commonly used classifiers, some of which, but not all, can be found in the similar work in [8]:

1. **Decision Tree** is a non-parametric supervised learning algorithm used for classification that learns from data to approximate a hierarchical three with a simple decision rule in each node. This tree can be seen as a line in a multidimensional plane that splits the training data into two areas (positive and negative examples).
2. **Logistic Regression** is a linear model for classification. It is based on optimizing the parameter of a logistic function to output the probability of a given example to be positive or negative
3. **Stochastic Gradient Descent** is an optimizer that uses gradient descent and a loss function to estimate the regression function. In this case, the loss function to be used will be a linear SVM.
4. **Ridge Classifier** is a regressor that first converts the binary label in {-1, 1} and then treats the problem as a regression task, optimizing a different function than SGD and Logistic Regression.
5. **Naive Bayes** is an algorithm that leverages the Bayes' theorem with the assumption of independence between any of the pair of features (a Naive assumption)
6. **Support Vector Classificator** is a classifier that finds the best boundary in a hyperplane between the two classes expanding the dimensions of the feature vectors.

[18]:
```
class RidgeClassifierWithProba(RidgeClassifier):
    def predict_proba(self, X):
        d = self.decision_function(X)
        d_2d = np.c_[-d, d]
```

```python
        return softmax(d_2d)

#Baseline models
models= [tree.DecisionTreeClassifier(random_state=42),
         LogisticRegression(random_state=42),
         SGDClassifier(max_iter=1000, tol=1e-3, loss='modified_huber',
 ↪random_state=42),
         RidgeClassifierWithProba(random_state=42),
         MultinomialNB(),
         SVC(probability=True, random_state=42)
        ]
model_names = [
    "Decision Tree",
    "Logistic Regression",
    "Stocasthic Gradient Descent",
    "Ridge",
    "Naive Bayes",
    "SVC"
]

#Columns of the evaluation metric dataframes
metric_df_columns = [
    'Dataset',
    'Model',
    'F1-Score',
    'Precision',
    'Recall',
    'Accuracy'
]
```

[7]:
```python
def model_metrics(ds_name, model_name, y, yhat):
    """
    Takes in a model name, ground truth and predictions
    and outputs an array of metrics

    Parameters
    ----------
    ds_name : string
        Dataset name
    model_name : string
        String name of the model
    y : list
        Ground truth
    yhat: list
        Model predictions

```

```
    Returns
    -------
    metric : list
        List containing the accuracy, f1-score,
        precision and recall of the model

    """
    return [
        ds_name,
        model_name,
        f1_score(y, yhat),
        precision_score(y, yhat),
        recall_score(y, yhat),
        accuracy_score(y, yhat)
    ]
```

[8]:
```
# Test train split for the 3 dataset, size is 20-80
i_X_train, i_X_test, i_y_train, i_y_test = train_test_split(i_X, i_y,␣
 ↪test_size=0.2, random_state=42)
l_X_train, l_X_test, l_y_train, l_y_test = train_test_split(l_X, l_y,␣
 ↪test_size=0.2, random_state=42)
t_X_train, t_X_test, t_y_train, t_y_test = train_test_split(t_X, t_y,␣
 ↪test_size=0.2, random_state=42)
```

[9]:
```
i_scores = []
l_scores = []
t_scores = []

#Evaluating baseline models and storing scores (can take minutes depending on␣
 ↪the machine)
for i,model in enumerate(models):
    # ISOT dataset evaluation
    print('Fitting model ', model_names[i], 'for Fake News dataset...')
    model.fit(i_X_train, i_y_train)
    yhat = model.predict(i_X_test)
    i_scores.append(model_metrics('ISOT', model_names[i], i_y_test, yhat))
    # LIAR dataset evaluation
    print('Fitting model ', model_names[i], 'for Liar dataset...')
    model.fit(l_X_train, l_y_train)
    yhat = model.predict(l_X_test)
    l_scores.append(model_metrics('LIAR', model_names[i], l_y_test, yhat))
    #TOTAL dataset evaluation
    print('Fitting model ', model_names[i], 'for Total dataset...')
    model.fit(t_X_train, t_y_train)
    yhat = model.predict(t_X_test)
    t_scores.append(model_metrics('TOTAL', model_names[i], t_y_test, yhat))
```

```
Fitting model  Decision Tree for Fake News dataset…
Fitting model  Decision Tree for Liar dataset…
Fitting model  Decision Tree for Total dataset…
Fitting model  Logistic Regression for Fake News dataset…
Fitting model  Logistic Regression for Liar dataset…
Fitting model  Logistic Regression for Total dataset…
Fitting model  Stocasthic Gradient Descent for Fake News dataset…
Fitting model  Stocasthic Gradient Descent for Liar dataset…
Fitting model  Stocasthic Gradient Descent for Total dataset…
Fitting model  Ridge for Fake News dataset…
Fitting model  Ridge for Liar dataset…
Fitting model  Ridge for Total dataset…
Fitting model  Naive Bayes for Fake News dataset…
Fitting model  Naive Bayes for Liar dataset…
Fitting model  Naive Bayes for Total dataset…
Fitting model  SVC for Fake News dataset…
Fitting model  SVC for Liar dataset…
Fitting model  SVC for Total dataset…
```

[10]:

```
i_scores_df = pd.DataFrame(i_scores, columns= metric_df_columns)
l_scores_df = pd.DataFrame(l_scores, columns= metric_df_columns)
t_scores_df = pd.DataFrame(t_scores, columns= metric_df_columns)

baseline_i_scores = i_scores.copy()
baseline_l_scores = l_scores.copy()
baseline_t_scores = t_scores.copy()
```

0.2.3 2.3 Classification Approach

The classificators that we'll be building are three different flavors of ensembles:

1. **Hard Blending Ensemble**, is a type of ensemble algorithm under the stacking generalization. There's is a meta-model that is created using the predictions of the base models, then the "blender" is fitted using a classificator model and the final prediction are the prediction of the blender model. In our case, the classificator of choice is the Logistic Regressor. The difference with the stacking generalization is that the meta-model is trained using a holdout dataset instead of k-fold cross-validation. The functions that implement this model are the "fit_ensemble" and the "predict_ensemble" taken from the beautiful article in [10] with some minor modification.
2. **Soft Blending Ensemble**, is the same as the hard, the difference is that the meta-model is composed of the probability of the given class instead of the actual prediction. It is implemented with the same functions as above with the difference in the hard parameter
3. **Soft Voting Ensemble**, is a simple voting ensemble in which the majority vote of the base models is the model prediction. It is soft because the base model's votes are weighted with the accuracy score of the models themselves on the validation set. The implementation is from sklearn but there's the "evaluate_models" function that is used to calculate the accuracies across the base models to be used as weights for the vote

The approach is similar to [9], with different ensemble methods and base models.

[11]:
```python
def fit_ensemble(models, X_train, X_val, y_train, y_val, hard=True):
    """
    Takes in a list of models, train and validation splits and
    performs the fitting of a blending ensemble with Logistic
    Regression (hard or soft).

    Parameters
    ----------
    models : list
        List of models
    X_train : list
        Training features
    X_val: list
        Validation features
    y_train : list
        Training labels
    y_val : list
        Validation labels
    hard : boolean
        Hard blending or soft blending
    Returns
    -------
    blender : object
        The fitted blender model to use for predictions

    """
    # fit all models on the training set and predict on hold out set
    meta_X = list()
    for model in models:
        # fit in training set
        model.fit(X_train, y_train)
        # predict on hold out set
        yhat = model.predict(X_val) if hard else model.predict_proba(X_val)
        # reshape predictions into a matrix with one column
        if hard:
            yhat = yhat.reshape(len(yhat), 1)
        # store predictions as input for blending
        meta_X.append(yhat)
    # create 2d array from predictions, each set is an input feature
    meta_X = hstack(meta_X)
    # define blending model
    blender = LogisticRegression()
    # fit on predictions from base models
    blender.fit(meta_X, y_val)
    return blender
```

[12]:
```python
# make a prediction with the blending ensemble
def predict_ensemble(models, blender, X_test, hard=True):
    """
    Takes in a list of models, a fitted blender, test features
    and performs the prediction of the labels with the fitted
    blender.

    Parameters
    ----------
    models : list
        List of models
    blender : object
        Fitted blender
    X_test : list
        Test features
    hard : boolean
        Hard blending or soft blending
    Returns
    -------
    predictions : list
        The predicted labels

    """
    # make predictions with base models
    meta_X = list()
    for model in models:
        # predict with base model
        yhat = model.predict(X_test) if hard else model.predict_proba(X_test)
        # reshape predictions into a matrix with one column
        if hard:
            yhat = yhat.reshape(len(yhat), 1)
        # store prediction
        meta_X.append(yhat)
    # create 2d array from predictions, each set is an input feature
    meta_X = hstack(meta_X)
    # predict
    return blender.predict(meta_X)
```

[13]:
```python
# evaluate each base model
def evaluate_models(models, X_train, X_val, y_train, y_val):
    """
    Takes in a list of models, train and validation splits and
    returns the accuracy of the models as an array

    Parameters
    ----------
    models : list
```

```
        List of models
    X_train : list
        Training features
    X_val: list
        Validation features
    y_train : list
        Training labels
    y_val : list
        Validation labels
    Returns
    -------
    accuracies : list
        List of accuracies of the models

    """
    # fit and evaluate the models
    scores = list()
    for model in models:
        # fit the model
        model.fit(X_train, y_train)
        # evaluate the model
        yhat = model.predict(X_val)
        acc = accuracy_score(y_val, yhat)
        # store the performance
        scores.append(acc)
    # report model performance
    return scores
```

[14]:
```
# split training set into train and validation sets ISOT dataset
X_train, X_val, y_train, y_val = train_test_split(i_X_train, i_y_train,␣
↪test_size=0.33, random_state=42)

#Scores backup in case evaluation runs multiple times
i_scores = baseline_i_scores.copy()

#Fitting and evaluating Hard Blender Ensemble
blender = fit_ensemble(models, X_train, X_val, y_train, y_val)
yhat = predict_ensemble(models, blender, i_X_test)
i_scores.append(model_metrics('ISOT', 'Hard Blender', i_y_test, yhat))

#Fitting and evaluationg Soft Blender Ensemble
blender = fit_ensemble(models, X_train, X_val, y_train, y_val, False)
yhat = predict_ensemble(models, blender, i_X_test, False)
i_scores.append(model_metrics('ISOT', 'Soft Blender', i_y_test, yhat))

#Fitting and evaluating Soft Voting Ensemble
accuracies = evaluate_models(models, X_train, X_val, y_train, y_val)
```

```
ensemble = VotingClassifier(estimators=list(zip(model_names, models)),␣
 ↪voting='soft', weights=accuracies)
ensemble.fit(X_train, y_train)
yhat = ensemble.predict(i_X_test)
i_scores.append(model_metrics('ISOT', 'Soft Voting Ensemble', i_y_test, yhat))

i_scores_df = pd.DataFrame(i_scores, columns= metric_df_columns)
```

[15]:
```
# split training set into train and validation sets LIAR dataset
X_train, X_val, y_train, y_val = train_test_split(l_X_train, l_y_train,␣
 ↪test_size=0.33, random_state=42)

#Scores backup in case evaluation runs multiple times
l_scores = baseline_l_scores.copy()

#Fitting and evaluating Hard Blender Ensemble
blender = fit_ensemble(models, X_train, X_val, y_train, y_val)
yhat = predict_ensemble(models, blender, l_X_test)
l_scores.append(model_metrics('LIAR', 'Hard Blender', l_y_test, yhat))

#Fitting and evaluationg Soft Blender Ensemble
blender = fit_ensemble(models, X_train, X_val, y_train, y_val, False)
yhat = predict_ensemble(models, blender, l_X_test, False)
l_scores.append(model_metrics('LIAR', 'Soft Blender', l_y_test, yhat))

#Fitting and evaluating Soft Voting Ensemble
accuracies = evaluate_models(models, X_train, X_val, y_train, y_val)
ensemble = VotingClassifier(estimators=list(zip(model_names, models)),␣
 ↪voting='soft', weights=accuracies)
ensemble.fit(X_train, y_train)
yhat = ensemble.predict(l_X_test)
l_scores.append(model_metrics('LIAR', 'Soft Voting Ensemble', l_y_test, yhat))

l_scores_df = pd.DataFrame(l_scores, columns= metric_df_columns)
```

[16]:
```
# split training set into train and validation sets TOTAL dataset
X_train, X_val, y_train, y_val = train_test_split(t_X_train, t_y_train,␣
 ↪test_size=0.33, random_state=42)

#Scores backup in case evaluation runs multiple times
t_scores = baseline_t_scores.copy()

#Fitting and evaluating Hard Blender Ensemble
blender = fit_ensemble(models, X_train, X_val, y_train, y_val)
yhat = predict_ensemble(models, blender, t_X_test)
t_scores.append(model_metrics('TOTAL', 'Hard Blender', t_y_test, yhat))
```

```python
#Fitting and evaluationg Soft Blender Ensemble
blender = fit_ensemble(models, X_train, X_val, y_train, y_val, False)
yhat = predict_ensemble(models, blender, t_X_test, False)
t_scores.append(model_metrics('TOTAL', 'Soft Blender', t_y_test, yhat))

#Fitting and evaluationg Soft Voting Ensemble
accuracies = evaluate_models(models, X_train, X_val, y_train, y_val)
ensemble = VotingClassifier(estimators=list(zip(model_names, models)),
 ↪voting='soft', weights=accuracies)
ensemble.fit(X_train, y_train)
yhat = ensemble.predict(t_X_test)
t_scores.append(model_metrics('TOTAL', 'Soft Voting Ensemble', t_y_test, yhat))

t_scores_df = pd.DataFrame(t_scores, columns= metric_df_columns)
```

0.3 3. Conclusion

0.3.1 3.1 Evaluation

Below is the report of the metrics evaluated for the six baseline models and the three ensembles.

For the **ISOT** dataset we have a clear winner, the probability blending ensemble algorithm performed best in 3 out of 4 metrics with a clear improvement in Accuracy, The Stochastic gradient descent was better in Precision, meaning that is less likely to give a false positive. The scores are very high indicating that the tf-idf representation of the text is a good feature selection method for this kind of long article.

For the **LIAR** dataset the scores are very low, slightly above 50% that is the same as a random classificator. The best algorithms were the Decision Tree and the Soft Voting Ensemble with a slightly better score overall for the voting ensemble. The Decision Tree has a better Recall, meaning that is less likely to give a false negative. The statements in the LIAR dataset are very short, probably the tf-idf matrix is not very good at capturing the patterns in the text to label fake news.

For the **TOTAL** dataset the best algorithm is the Soft Blender, with the Naive Bayes having a better precision and the SVC having better accuracy. The Soft Blender is more balanced, having a higher f1-score.

	Dataset	Model	F1-Score	Precision	Recall	Accuracy
0	ISOT	Decision Tree	0.916667	0.913495	0.919861	0.92
1	ISOT	Logistic Regression	0.819149	0.833935	0.804878	0.83
2	ISOT	Stocasthic Gradient Descent	0.894737	0.971429	0.829268	0.906667
3	ISOT	Ridge	0.868651	0.873239	0.864111	0.875
4	ISOT	Naive Bayes	0.640553	0.945578	0.484321	0.74
5	ISOT	SVC	0.881834	0.892857	0.87108	0.888333
6	ISOT	**Hard Blender**	0.928058	0.959108	0.898955	0.933333
7	ISOT	**Soft Blender**	0.955017	0.948454	0.961672	0.956667
8	ISOT	**Soft Voting Ensemble**	0.935652	0.934028	0.937282	0.938333

	Dataset	Model	F1-Score	Precision	Recall	Accuracy
0	LIAR	Decision Tree	**0.46461**	0.465455	**0.463768**	0.508333
1	LIAR	Logistic Regression	0.364066	0.52381	0.278986	0.551667
2	LIAR	Stocasthic Gradient Descent	0.00719424	0.5	0.00362319	0.54
3	LIAR	Ridge	0.421053	0.533333	0.347826	0.56
4	LIAR	Naive Bayes	0.0070922	0.166667	0.00362319	0.533333
5	LIAR	SVC	0.335	0.540323	0.242754	0.556667
6	LIAR	**Hard Blender**	0.340852	0.552846	0.246377	0.561667
7	LIAR	**Soft Blender**	0.421053	0.571429	0.333333	0.578333
8	LIAR	**Soft Voting Ensemble**	0.46087	**0.576087**	0.384058	**0.586667**
0	TOTAL	Decision Tree	0.67288	0.667283	0.678571	0.7075
1	TOTAL	Logistic Regression	0.657084	0.723982	0.601504	0.721667
2	TOTAL	Stocasthic Gradient Descent	0.532676	0.774194	0.406015	0.684167
3	TOTAL	Ridge	0.692913	0.727273	0.661654	0.74
4	TOTAL	Naive Bayes	0.371345	**0.835526**	0.238722	0.641667
5	TOTAL	SVC	0.694	0.741453	0.652256	**0.745**
6	TOTAL	**Hard Blender**	0.687627	0.746696	0.637218	0.743333
7	TOTAL	**Soft Blender**	**0.703669**	0.704331	**0.703008**	0.7375
8	TOTAL	**Soft Voting Ensemble**	0.684418	0.719917	0.652256	0.733333

[17]:
```
scores_df = i_scores_df.append(l_scores_df).append(t_scores_df)
scores_df
```

[17]:
```
  Dataset                        Model  F1-Score  Precision    Recall  \
0    ISOT                Decision Tree  0.916667   0.913495  0.919861
1    ISOT          Logistic Regression  0.819149   0.833935  0.804878
2    ISOT  Stocasthic Gradient Descent  0.894737   0.971429  0.829268
3    ISOT                        Ridge  0.868651   0.873239  0.864111
4    ISOT                  Naive Bayes  0.640553   0.945578  0.484321
5    ISOT                          SVC  0.881834   0.892857  0.871080
6    ISOT                 Hard Blender  0.928058   0.959108  0.898955
7    ISOT                 Soft Blender  0.955017   0.948454  0.961672
8    ISOT         Soft Voting Ensemble  0.935652   0.934028  0.937282
0    LIAR                Decision Tree  0.464610   0.465455  0.463768
1    LIAR          Logistic Regression  0.364066   0.523810  0.278986
2    LIAR  Stocasthic Gradient Descent  0.007194   0.500000  0.003623
3    LIAR                        Ridge  0.421053   0.533333  0.347826
4    LIAR                  Naive Bayes  0.007092   0.166667  0.003623
5    LIAR                          SVC  0.335000   0.540323  0.242754
6    LIAR                 Hard Blender  0.340852   0.552846  0.246377
7    LIAR                 Soft Blender  0.416476   0.565217  0.329710
8    LIAR         Soft Voting Ensemble  0.450766   0.569061  0.373188
0   TOTAL                Decision Tree  0.672880   0.667283  0.678571
1   TOTAL          Logistic Regression  0.657084   0.723982  0.601504
2   TOTAL  Stocasthic Gradient Descent  0.532676   0.774194  0.406015
```

3	TOTAL	Ridge	0.692913	0.727273	0.661654
4	TOTAL	Naive Bayes	0.371345	0.835526	0.238722
5	TOTAL	SVC	0.694000	0.741453	0.652256
6	TOTAL	Hard Blender	0.687627	0.746696	0.637218
7	TOTAL	Soft Blender	0.703669	0.704331	0.703008
8	TOTAL	Soft Voting Ensemble	0.685714	0.720497	0.654135

	Accuracy
0	0.920000
1	0.830000
2	0.906667
3	0.875000
4	0.740000
5	0.888333
6	0.933333
7	0.956667
8	0.938333
0	0.508333
1	0.551667
2	0.540000
3	0.560000
4	0.533333
5	0.556667
6	0.561667
7	0.575000
8	0.581667
0	0.707500
1	0.721667
2	0.684167
3	0.740000
4	0.641667
5	0.745000
6	0.743333
7	0.737500
8	0.734167

0.3.2 3.2 Summary and conclusions

The analysis has presented 6 commonly used algorithms for classification and used them to create an ensemble with three different methodologies. The objective of the paper was to create an ensemble that outperforms the models it is composed of. The objective has been partially met for the LIAR dataset and fully met for the ISOT dataset. The secondary objective was to find out which of the three methodologies to train an ensemble is the best and the answer seems arguable to be the probability blending ensemble. Another reflection must be done on the very low accuracy performance of the algorithms for the LIAR dataset, seems that the tf-idf matrix did not capture enough information from the statements to be anywhere accurate. In this paper, we focused on the text statement ignoring the other interesting columns presented in the dataset. The Non-text attributes of the dataset could be a key factor in training a successful algorithm, the speaker of the statement and the affiliation party can be a key indicators for tweaking the probability of a statement being fake news.

This work is further validation of the work presented in [8] and [9], an additional quantitative proof that the ensembles showed an overall better score as compared to the weak learners.

Further extensions on the idea presented in this work could be the hyper-parameter optimization for the learners with cross-fold validation, one of the hyper-parameter being the ensemble components (different sets of classificators). This can be a very slow process if using the entire ISOT dataset for example, and can probably increase the overfitting of the final ensemble.

0.3.3 References

[1] https://en.wikipedia.org/wiki/Fake_news

[2] Y. Chen, N. J. Conroy, and V. L. Rubin, “News in an online world: The need for an automatic crap detector,” Proceedings of the Association for Information Science and Technology, vol. 52, no. 1, pp. 1–4, 2015.

[3] Parikh, S.B. & Atrey, P.K. 2018, “Media-Rich Fake News Detection: A Survey”, IEEE, , pp. 436.

[4] W. Y. Wang, “ ” liar, liar pants on fire”: A new benchmark dataset for fake news detection,” arXiv preprint arXiv:1705.00648, 2017

[5] Ahmed H, Traore I, Saad S. “Detecting opinion spams and fake news using text classification”, Journal of Security and Privacy, Volume 1, Issue 1, Wiley, January/February 2018.

[6] Ahmed H, Traore I, Saad S. (2017) “Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques. In: Traore I., Woungang I., Awad A. (eds) Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments. ISDDC 2017. Lecture Notes in Computer Science, vol 10618. Springer, Cham (pp. 127-138).

[7] https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset

[8] Arvin Hansrajh, Timothy T. Adeliyi, Jeanette Wing, “Detection of Online Fake News Using Blending Ensemble Learning”, Scientific Programming, vol. 2021, Article ID 3434458, 10 pages, 2021. https://doi.org/10.1155/2021/3434458

[9] Iftikhar Ahmad, Muhammad Yousaf, Suhail Yousaf, Muhammad Ovais Ahmad, “Fake News Detection Using Machine Learning Ensemble Methods”, Complexity, vol. 2020, Article ID 8885861, 11 pages, 2020. https://doi.org/10.1155/2020/8885861

[10] https://machinelearningmastery.com/blending-ensemble-machine-learning-with-python/