A Twitter-Based Climate Change Stance Classification Pipeline

Jiarong Ye
College of Engineering
jxy225@psu.edu

Yuan Meng
College of Information Science and Technology
xxx@psu.edu

Abstract

The goal of the mini project is to get a hands-on personal experience regarding the construction, interpretation, refinement, deployment, and evaluation of a twitter-based stance classification pipeline.

1 Introduction

[to be completed ...]

2 Data Preparation

[to be completed ...]

3 Feature Extraction

[to be completed ...]

4 Model Construction

Using the set of provided tweets (which have been tagged both for Relevant and for Stance) to construct two Decision Tree-based predictive models predicting whether a tweet is supportive or lack of support for the topic.

4.1 Baseline Model

4.1.1 Parameters

Listing 1: parameters of baseline model

```
* CountVector
        * token_pattern='(([#@]|[0-9]|[a-z]|[A-Z])+)'
        * analyzer = 'word'
        * min_df = 3
* DecisionTree
        * criterion='entropy'
        * max_depth = 7
        * min_samples_leaf = 2
```

4.1.2 Dataset

DS1 [Descriptions to be completed ...]

4.2 Improved Model

The improvements of the following parameters of the Baseline Model were implemented:

4.2.1 Token Pattern

Listing 2: Adjustment of token pattern to improve the model's interpretability

```
* Baseline Model
        * token_pattern="(([#@]|[0-9]|[a-z]|[A-Z])+)"

* Improved Model
        * token_pattern="(?!RT|rt|\d+)[@#]*[\w\'_-]{2,100}"
```

4.2.2 Max Depth

1-15 [Details to be completed ...]

4.2.3 Stop Words

Option 1:

default ('english')

Table 1: F1 Score

Max Depth	Test Set	Train Set
1	0.690	0.597
2	0.688	0.682
3	0.684	0.688
4	0.693	0.697
5	0.693	0.703
6	0.693	0.706
7	0.694	0.711
8	0.682	0.714
9	0.704	0.718
10	0.693	0.719
11	0.677	0.723
12	0.688	0.728
13	0.683	0.733
14	0.677	0.736
15	0.672	0.735

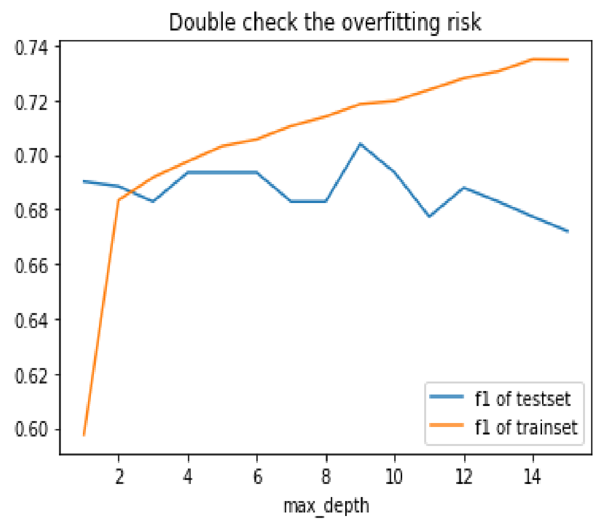

Figure 1: F1 score of training and testing set

option 2:

self-defined: 'a', 'an', 'the', 'it', 'is', 'are', 'be', 'of', 'this', 'that', 'RT', 'rt','https'

Table 2: F1 Score

Max Depth	Test Set	Train Set
1	0.479	0.605
2	0.648	0.683
3	0.672	0.698
4	0.660	0.702
5	0.637	0.706
6	0.631	0.714
7	0.649	0.721
8	0.650	0.721
9	0.631	0.724
10	0.693	0.719
11	0.620	0.731
12	0.631	0.739
13	0.621	0.747
14	0.596	0.749
15	0.614	0.748

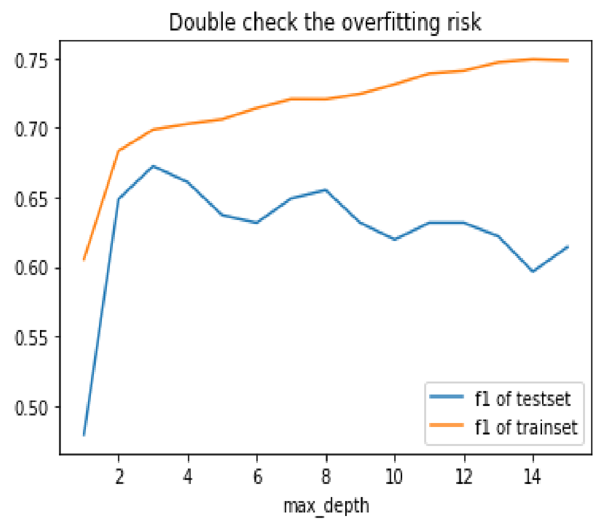

Figure 2: F1 score of training and testing set

option 3:

The top 30 most common words in DS1+DS2 tweets (exclude semantic sensitive words like **climate** and **change**)

Listing 3: Get the words with the highest frequency

```
stop_w = [i[0] for i in Counter([word for sentence in X for word in sentence.split()
                if 'climate' not in word.lower() and 'change' not in word.lower()
                and word.isalpha()
                and len(word)>1]).most_common()[:30]]

output:
        ['the', 'to', 'RT', 'of', 'is', 'and','in', 'that', 'on', 'for', 'are', 'you',
        'we', 'The', 'it', 'this', 'about', 'be', 'by', 'have', 'not', 'will', 'our',
        'from', 'as', 'with', 'can', 'all', 'We', 'do']
```

Table 3: F1 Score

Max Depth	Test Set	Train Set
1	0.647	0.599
2	0.708	0.683
3	0.690	0.693
4	0.700	0.699
5	0.701	0.707
6	0.701	0.712
7	0.684	0.717
8	0.690	0.717
9	0.690	0.721
10	0.724	0.729
11	0.719	0.733
12	0.714	0.735
13	0.708	0.741
14	0.714	0.740
15	0.702	0.745

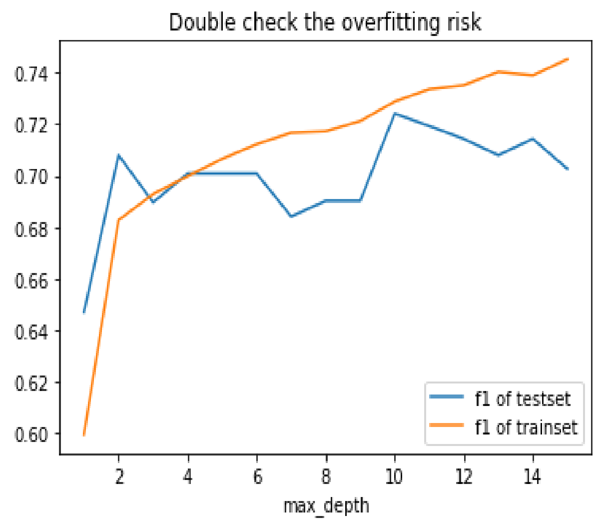

Figure 3: F1 score of training and testing set

4.3 Optimal Model

Listing 4: optimal parameters

```
* CountVector
        * token_pattern = "(?!RT|rt|\d+)[@#]*[\w\'_-]{2,100}"
        * analyzer = 'word'
        * stop_words = ['the', 'to', 'RT', 'of', 'is',  'and','in', 'that',
                        'on', 'for', 'are',      'you', 'we', 'The', 'it',
                        'this', 'about', 'be', 'by', 'have', 'not',
                        'will', 'our', 'from', 'as', 'with', 'can', 'all',
                        'We', 'do']
        * min_df = 3
* DecisionTree
        * criterion='entropy'
        * max_depth = 10
        * min_samples_leaf = 2
```

4.3.1 Dataset

DS2 [Descriptions [to be completed ...]]

5 Model Assessment

5.1 Baseline Model

Table 4: 5-fold output of decision tree with max_path=5, min_samples_leaf=2

Fold	Accuracy	Precision	Recall	F1 score
1	0.639535	0.649123	0.770833	0.704762
2	0.649805	0.680272	0.699301	0.689655
3	0.589844	0.613095	0.720280	0.662379
4	0.593750	0.610169	0.755245	0.675000
5	0.601562	0.627329	0.706294	0.664474

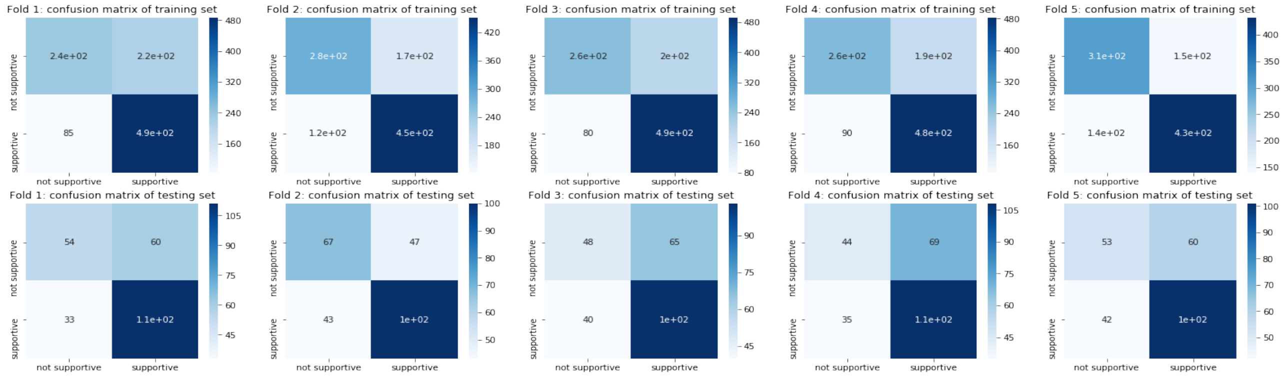

Figure 4: Confusion Matrix of training and testing set to observe overfitting

[to be completed ...]

5.2 Improved Model

Table 5: 5-fold output of decision tree with max_path=10, min_samples_leaf=2

Fold	Accuracy	Precision	Recall	F1 score
1	0.617594	0.611111	0.750853	0.673813
2	0.628366	0.627976	0.720137	0.670906
3	0.646320	0.639535	0.750853	0.690738
4	0.647482	0.639535	0.753425	0.691824
5	0.638489	0.631124	0.750000	0.685446

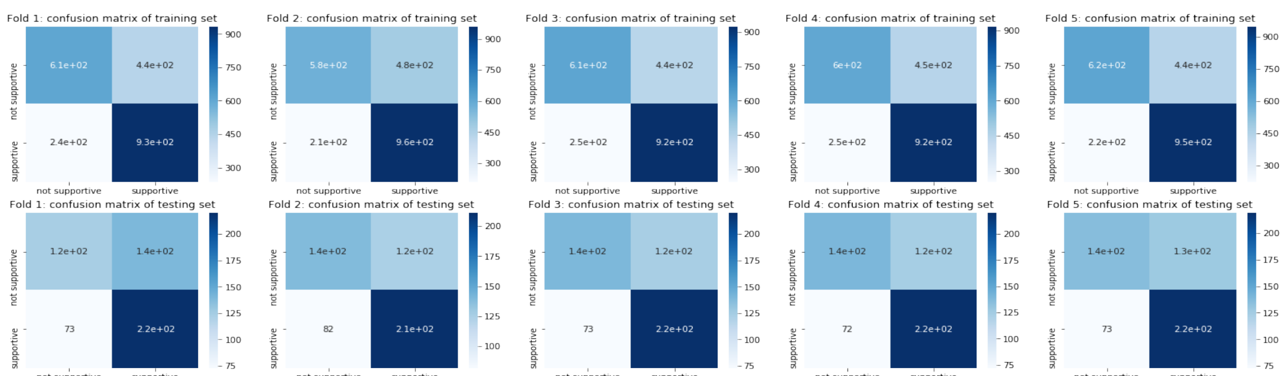

Figure 5: Confusion Matrix of training and testing set to observe overfitting

[to be completed ...]

6 Model Interpretation

6.1 Baseline Model

[Detailed discussion to be completed ...]

6.1.1 Fold 1

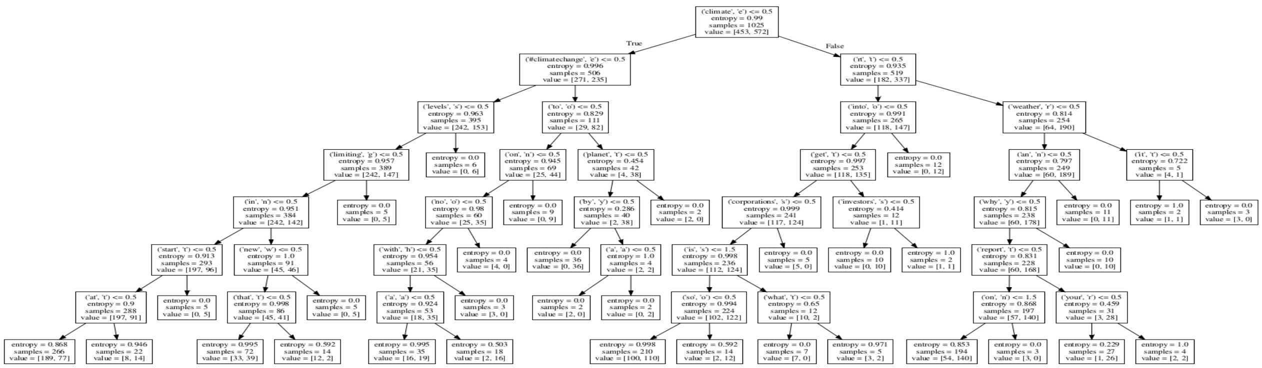

Figure 6: Decision Tree Fold 1

6.1.2 Fold 2

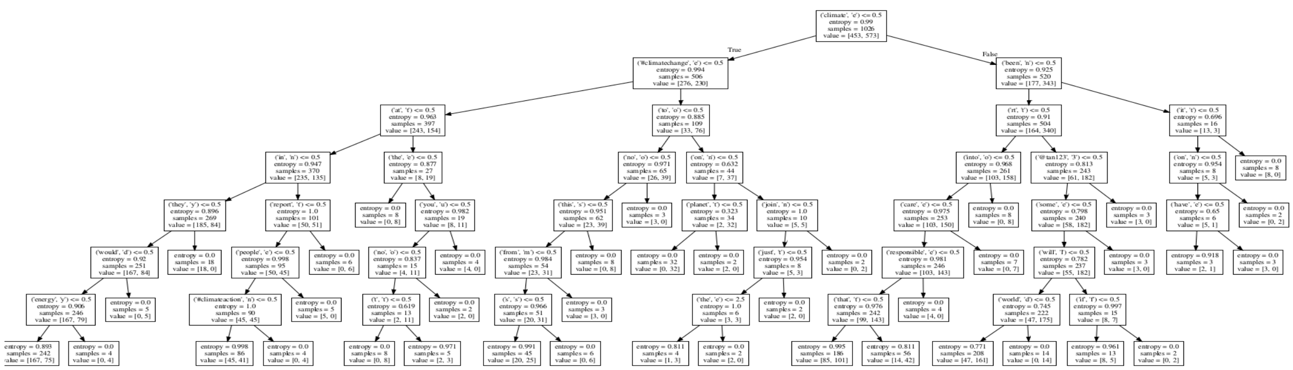

Figure 7: Decision Tree Fold 2

6.1.3 Fold 3

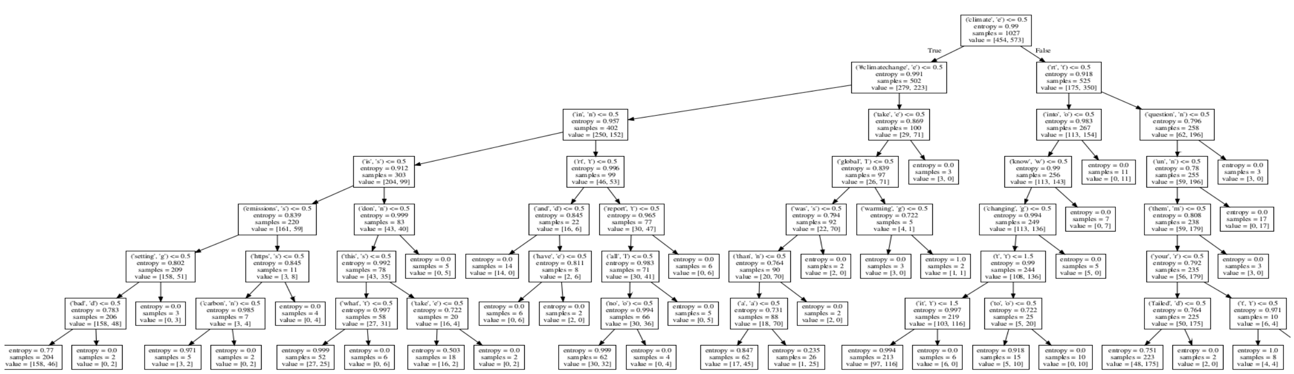

Figure 8: Decision Tree Fold 3

6.1.4 Fold 4

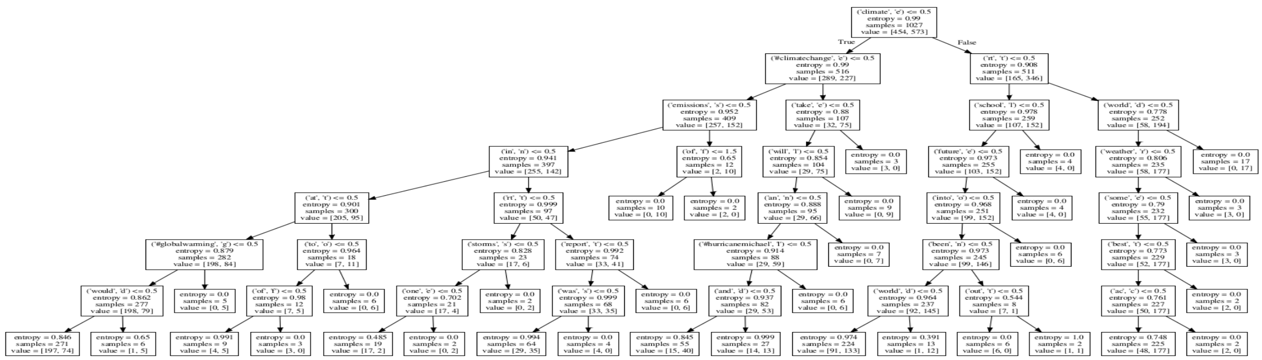

Figure 9: Decision Tree Fold 4

6.1.5 Fold 5

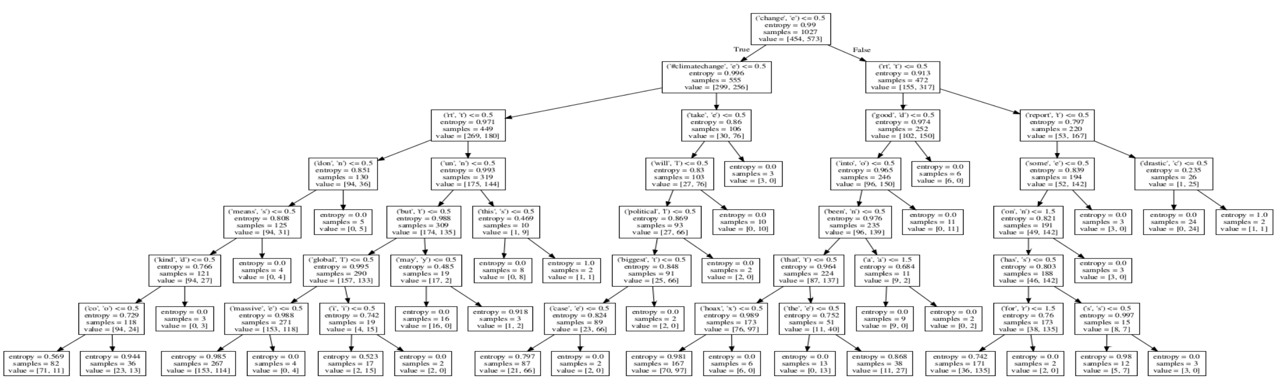

Figure 10: Decision Tree Fold 5

6.2 Improved Model

[Detailed discussion to be completed ...]

6.2.1 Fold 1

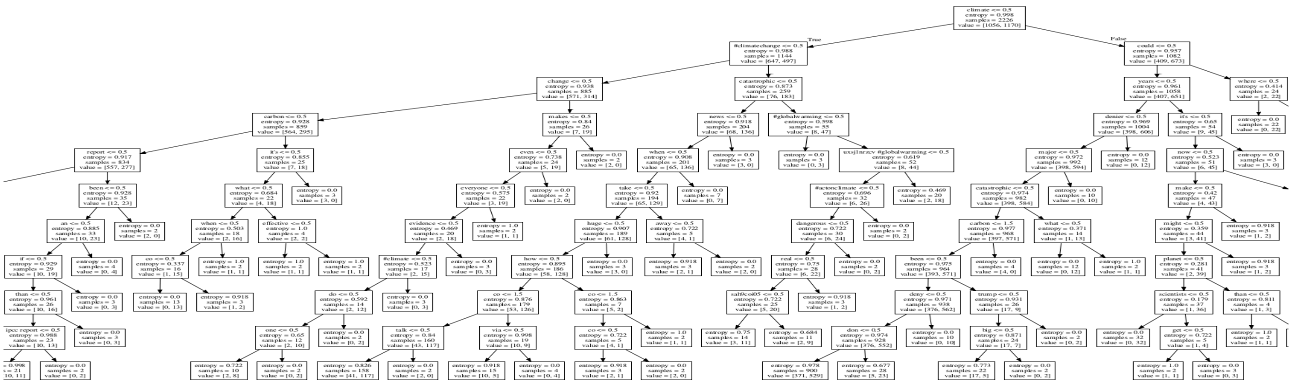

Figure 11: Decision Tree Fold 1 (partial)

6.2.2 Fold 2

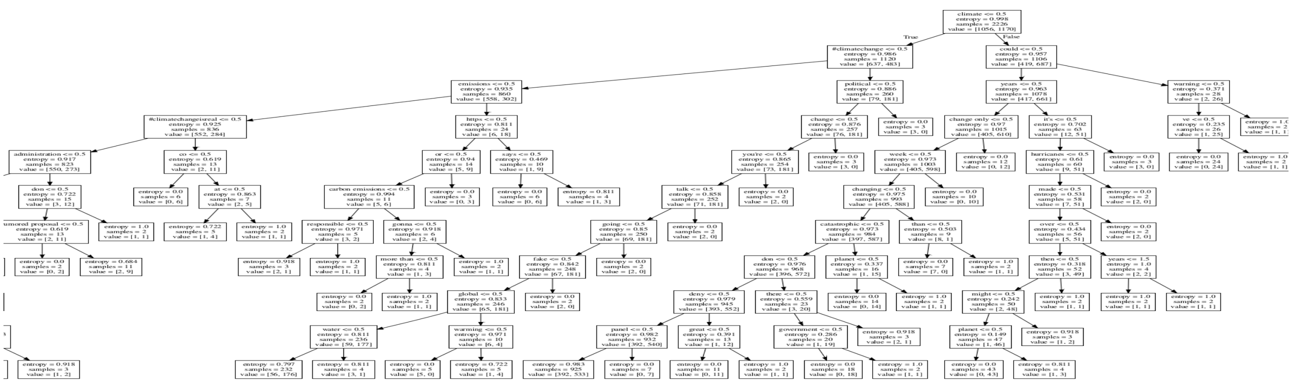

Figure 12: Decision Tree Fold 2 (partial)

6.2.3 Fold 3

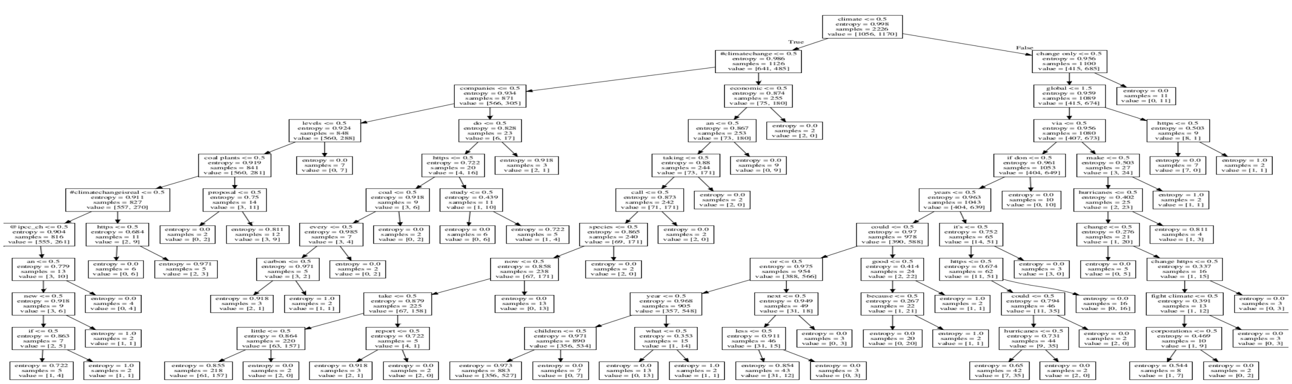

Figure 13: Decision Tree Fold 3 (partial)

6.2.4 Fold 4

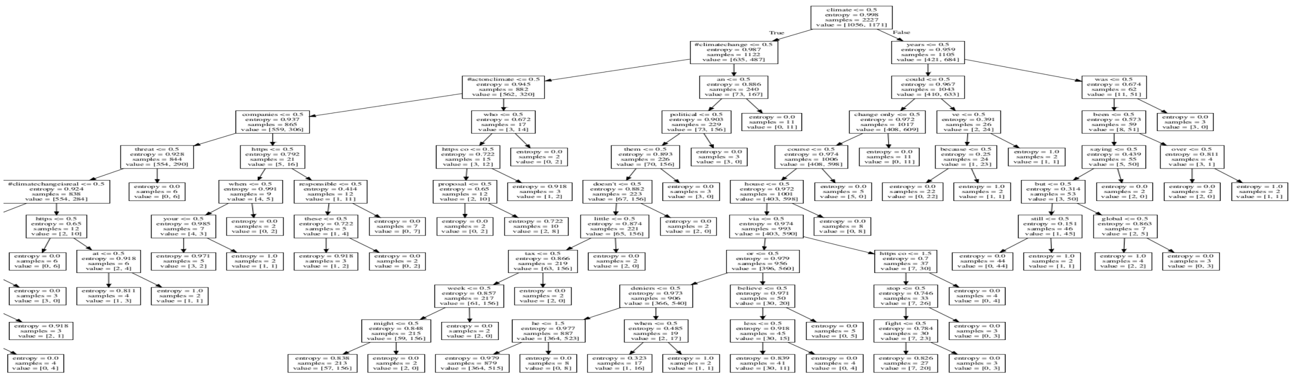

Figure 14: Decision Tree Fold 4 (partial)

6.2.5 Fold 5

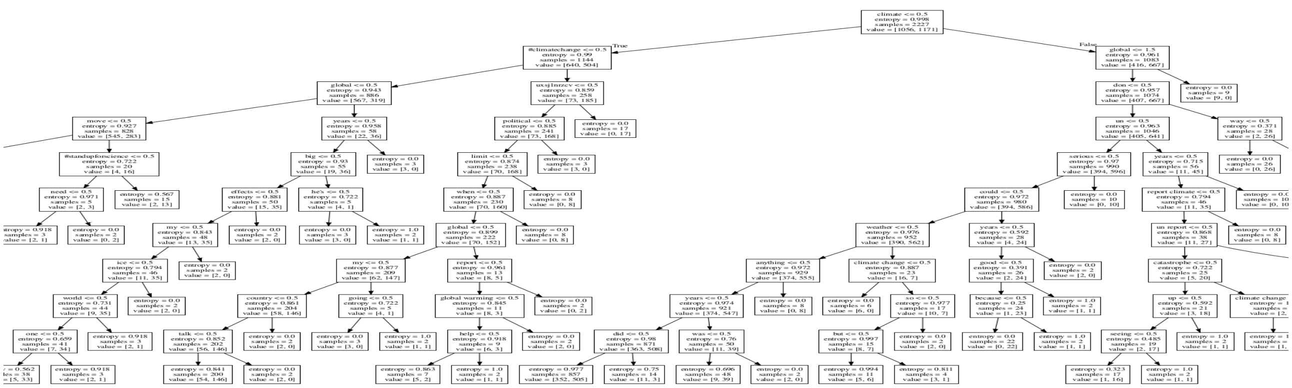

Figure 15: Decision Tree Fold 5 (partial)

7 Conclusion/Future Work

to be completed [1] ...

References

[1] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.

8 Appendix

A Python Code

Listing 5: import packages

```
import datascience as ds
from datascience import *
import numpy as np
from collections import Counter
from graphviz import Source
import pandas as pd
import seaborn as sns
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score,
                            accuracy_score, classification_report
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
from sklearn.externals import joblib
%matplotlib inline
```

Listing 6: Aggregate supportive/non-supportive Label

```
df = pd.read_csv(filepath, sep=',')

def aggregate_labels(df, agg_label, support_level):
        labels = df.columns[2:]
        for label in labels:
                for key, val in STANCE_VAL_DICT.items():
        df[label] = df[label].replace(key, val)
        df[agg_label] = df[df.columns[2:]].apply(lambda x: np.sum(x.values), axis=1)
        df[support_level] = df.iloc[:, -1].apply(lambda x: 1 if x>=1 else 0)
        df = df.drop(columns=df.columns[2:6])
        supportive_cnt = df.loc[df[support_level]==1, :].shape[0]
        unsupportive_cnt = df.loc[df[support_level]==0, :].shape[0]
        df.to_csv('Climate1SupportiveLevel.csv')

aggregate_labels(df, 'FinalLabel', 'SupportiveLabel')
```

Listing 7: Combine DS1 and DS2 and randomly extract 100 tweets for final testing

```
df1 = ds.Table.read_table('Climate1SupportiveLevel.csv', sep=',')
df2 = ds.Table.read_table('ClimateBalancedDS2.csv', sep=',')
df = df1.append(df2)
test_index = np.random.choice(df.num_rows, 100, replace=False)
train_val_index = [i for i in np.arange(df.num_rows) if i not in test_index]
test_data = df.take[test_index]
df = df.take[train_val_index]
X = list(df['Text'])
y = list(df['Support'])
test_X = list(test_data['Text'])
test_y = list(test_data['Support'])
```

Listing 8: Check whether the data distribution is balanced

```
def check(sentiment, index, note='training'):
        if sentiment==0:
                label = 'not supportive'
        else:
                label = 'supportive'
        print('There are {} '.format(df.take(index).where('Support',
                are.equal_to(sentiment)).size[0][0])+label+' tweets in the '+note+' set.')
```

Listing 9: Split data into training and validation dataset

```
def custom_split(train_index, test_index):
        trainingset = df.take(train_index)
        testingset = df.take(test_index)

        X_train= list(trainingset['Text'])
        y_train= list(trainingset['Support'])
        X_test= list(testingset['Text'])
        y_test= list(testingset['Support'])

        return X_train, X_test, y_train, y_test
```

Listing 10: Split data into training and validation dataset

```
def classifier(X_train, y_train, X_test, fold, max_depth, min_samples_leaf):
        clf = Pipeline(
                [('vect', CountVectorizer(token_pattern="(?!RT|rt|\d+)[@#]*[\w\'_-]{2,100}",
                        analyzer = 'word',
                        stop_words = 'english',
                        min_df = 4,
                        ngram_range=(1,2))),
                ('clf', DecisionTreeClassifier(criterion='entropy',
                        random_state = 100,
                        max_depth = max_depth,
                        min_samples_leaf = min_samples_leaf))
                ])
        clf.fit(X_train, y_train)
        feature_names = clf.named_steps['vect'].get_feature_names()
        try:
                dot_data = tree.export_graphviz(clf.named_steps['clf'], out_file=None,
                feature_names=feature_names)
                graph = Source(dot_data)
                graph.render('ClimateClassifier-Fold_{}'.format(fold))
        except Exception as e:
                print(e)
        predicted_y_train = clf.predict(X_train)
        predicted_y_test = clf.predict(X_test)
        # save as pickle
        joblib.dump(clf, 'ClimateTeam7PD2.pkl')
        return predicted_y_train, predicted_y_test
```

Listing 11: Evaluation Metrics

```
def eval_results(predicted_y_train, y_train, predicted_y_test, y_test, fold):
        accuracy_s = accuracy_score(y_test, predicted_y_test)
        precision_s = precision_score(y_test, predicted_y_test)
        recall_s = recall_score(y_test, predicted_y_test)
        f1_s = f1_score(y_test, predicted_y_test)
        cm_train = confusion_matrix(y_train, predicted_y_train)
        cm_test = confusion_matrix(y_test, predicted_y_test)

        print('Accuracy Score:', accuracy_s)
        print("Precision Score:", precision_s)
        print("Recall Score:", recall_s)
        print("f1 Score:", f1_s)
        print('confusion_matrix of training set is: \n', cm_train, '\n')
        print('confusion_matrix of testing set is: \n', cm_test, '\n')
        print(classification_report(y_test, predicted_y_test))

        classes = ['not supportive', 'supportive']
        plt.subplot(2, 5, fold)
        sns.heatmap(cm_train, annot=True, cmap='Blues', yticklabels=classes,
                                                        xticklabels=classes)
        plt.title('Fold {}: confusion matrix of training set'.format(fold))
```

```
        plt.subplot(2, 5, fold+5)
        sns.heatmap(cm_test, annot=True, cmap='Blues', yticklabels=classes,
                                                        xticklabels=classes)
        plt.title('Fold {}: confusion matrix of testing set'.format(fold))
        return accuracy_s, precision_s, recall_s, f1_s
```

Listing 12: Training with K-fold cross validation

```
def k_fold_evaluate(X, y, max_depth, min_samples_leaf, stop_words,
                              print_eval=True, overfit_risk=False):
        # initialization
        accuracy = []
        precision = []
        recall=[]
        f1 = []
        fold = 1
        skf = StratifiedKFold(n_splits=5, random_state=1, shuffle= True)

        # build model and collect results
        for val_index, test_index in skf.split(X, y):
                X_train, X_val, y_train, y_val = custom_split(val_index, test_index)

                predicted_y_train, predicted_y_val = classifier(
                                                        X_train=X_train,
                                                        y_train=y_train,
                                                        X_test=X_val, fold=fold,
                                                        max_depth = max_depth,
                                                        min_samples_leaf = min_samples_leaf,
                                                        stop_words = stop_words,
                                                        overfit_risk=overfit_risk)
                metrics_df={}
                if print_eval:
                        print('\nFold: {}'.format(fold))
                        accuracy_s, precision_s, recall_s, f1_s =
                         eval_results(predicted_y_train, y_train, predicted_y_val, y_val)

                        accuracy.append(accuracy_s)
                        precision.append(precision_s)
                        recall.append(recall_s)
                        f1.append(f1_s)

                        metrics_df = pd.DataFrame(
                        {
                                'accuracy': accuracy,
                                'precision': precision,
                                'recall':recall,
                                'f1':f1
                        })
                fold += 1
        return metrics_df
```

Listing 13: Testing

```
f1_lst_test = []
f1_lst_train = []
for d in range(1, 15):
        k_fold_evaluate(X, y, max_depth=d, min_samples_leaf=2, stop_words='english',
                        print_eval=False, overfit_risk=True)
        clf_tmp = joblib.load('ClimateTeam7PD2_maxdepth{}.pkl'.format(d))
        print('maxdepth=', d)
        # test
        y_pred = clf_tmp.predict(test_X)
        print('test f1')
        print(f1_score(y_pred=y_pred, y_true=test_y))
```

```
        f1_lst_test1.append(f1_score(y_pred=y_pred, y_true=test_y))
        # train_val
        y_pred = clf_tmp.predict(X)
        print('train f1')
        print(f1_score(y_pred=y_pred, y_true=y))
        f1_lst_train1.append(f1_score(y_pred=y_pred, y_true=y))
```

Listing 14: Final Result

```
k_fold_evaluate(X, y, max_depth=10, min_samples_leaf=2,
stop_words=stop_w,
print_eval=True, overfit_risk=False)
clf2 = joblib.load('ClimateTeam7PD2.pkl')
```