Getting Data Together into arrays

In [ ]:
```
# Import packages
import os, json, nltk
import numpy as np
from nltk.stem.snowball import EnglishStemmer
import matplotlib.pyplot as plt
```

The dataset has been downloaded already, but it needs to be imported from the JSON objects.

In [ ]:
```
#Importing Data
skip = True
title = []
text = []
url = []
labels = []

#Getting fake news
for x in os.walk("fakenewsnet_dataset/politifact/fake"):

    #Skipping first because it is the directory itself
    if skip:
        skip = False
        continue
    try:
        dir = x[0] + "/news content.json"
        dir = dir.replace("\\","/")
        o = json.load(open(dir))

        #If there is text, get title and text. If there are any errors move on
        if o["text"] != "":
            title.append(o["title"])
            text.append(o["text"])
            url.append(o["url"])
            labels.append(1)
    except Exception:
        continue

skip = True
for x in os.walk("fakenewsnet_dataset/politifact/real"):
    #Skipping first because it is the directory itself
    if skip:
        skip = False
        continue
    try:
        dir = x[0] + "/news content.json"
        dir = dir.replace("\\","/")
        o = json.load(open(dir))

        #If there is text, get title and text. If there are any errors move on
        if o["text"] != "":
            title.append(o["title"])
            text.append(o["text"])
            url.append(o["url"])
            labels.append(0)
    except Exception:
        continue
```

Preprocessing Data

The TFIDF vectorizer handles a lot of the preprocessing, but it does not stem words, so that will be done using Natural Language Toolkit.

In [ ]:
```
#Stemming using the english stemmer
stemmer = EnglishStemmer()
stemmed_text = text[:]
stemmed_title = title[:]

#Combined text and title here
for i in range(len(text)):
    stemmed_text[i] = ' '.join([stemmer.stem(word) for word in text[i].split(" ")])
    stemmed_title[i] = ' '.join([stemmer.stem(word) for word in title[i].split(" ")
    stemmed_text[i] = stemmed_title[i] + stemmed_text[i]
```

In [ ]:
```
#Loading data into Numpy arrays
stemmed_text = np.array(stemmed_text)
labels = np.array(labels)
```

Data Sentiment Analysis and Word Cloud

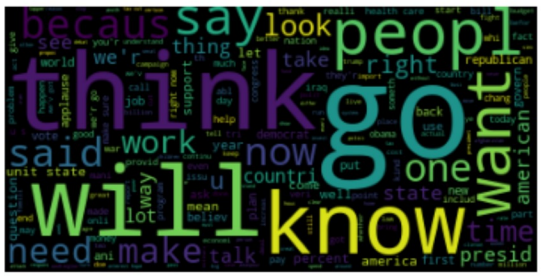
In [ ]:
```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')

text = np.array(text)

sia = SentimentIntensityAnalyzer()

#Sentiment analysis before stemming
fake = text[labels == 1]
real = text[labels == 0]
fake_sentiments = [sia.polarity_scores(sent)["compound"] for sent in fake]
real_sentiments = [sia.polarity_scores(sent)["compound"] for sent in real]

print("Compound score pre-stemming of fake news: %.5f" % np.mean(fake_sentiments))
print("Compound score pre-stemming of real news: %.5f" % np.mean(real_sentiments))

#Sentiment analysis after stemming
fake = stemmed_text[labels == 1]
real = stemmed_text[labels == 0]
fake_sentiments = [sia.polarity_scores(sent)["compound"] for sent in fake]
real_sentiments = [sia.polarity_scores(sent)["compound"] for sent in real]

print("Compound score post-stemming of fake news: %.5f" % np.mean(fake_sentiments))
print("Compound score post-stemming of real news: %.5f" % np.mean(real_sentiments))
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\Ishaan\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
Compound score pre-stemming of fake news: 0.02699
Compound score pre-stemming of real news: 0.48121
Compound score post-stemming of fake news: -0.08056
Compound score post-stemming of real news: 0.38306
```

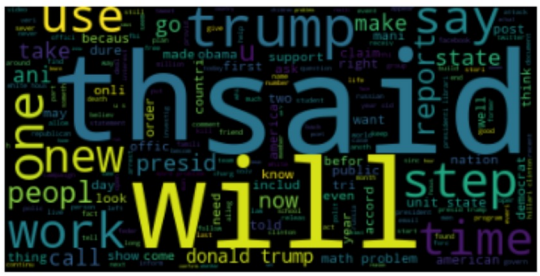
In [ ]:
```
#Word cloud generation
from wordcloud import WordCloud

#Putting all text together, it will handle tokenization
real = ' '.join(stemmed_text[labels == 0])
fake = ' '.join(stemmed_text[labels == 1])
wc_real = WordCloud().generate(text=real)
wc_fake = WordCloud().generate(text=fake)

#Plotting
fig = plt.figure(figsize=(8,5))
plt.imshow(wc_real)
plt.axis("off")
plt.title("Real News Wordcloud")

fig = plt.figure(figsize=(8,5))
plt.imshow(wc_fake)
plt.axis("off")
plt.title("Fake News Wordcloud")
```

Out[ ]:
```
Text(0.5, 1.0, 'Fake News Wordcloud')
```

Real News Wordcloud



Fake News Wordcloud



Tuning Hyperparameters

The hyperparameter to be tuned is the range of n-grams to use for the vectorizer. For example, a range of (1,3) means that the model is trained on uni-grams, bi-grams, and tri-grams. A range of (1,1) would mean the model only takes into account uni-grams, or just words by themselves.

In [ ]:
```
#Train test split
from sklearn.model_selection import train_test_split

seed = 42
X_train, X_test, y_train, y_test = train_test_split(stemmed_text,labels,test_size=0

#Sanity check
X_train.shape
```

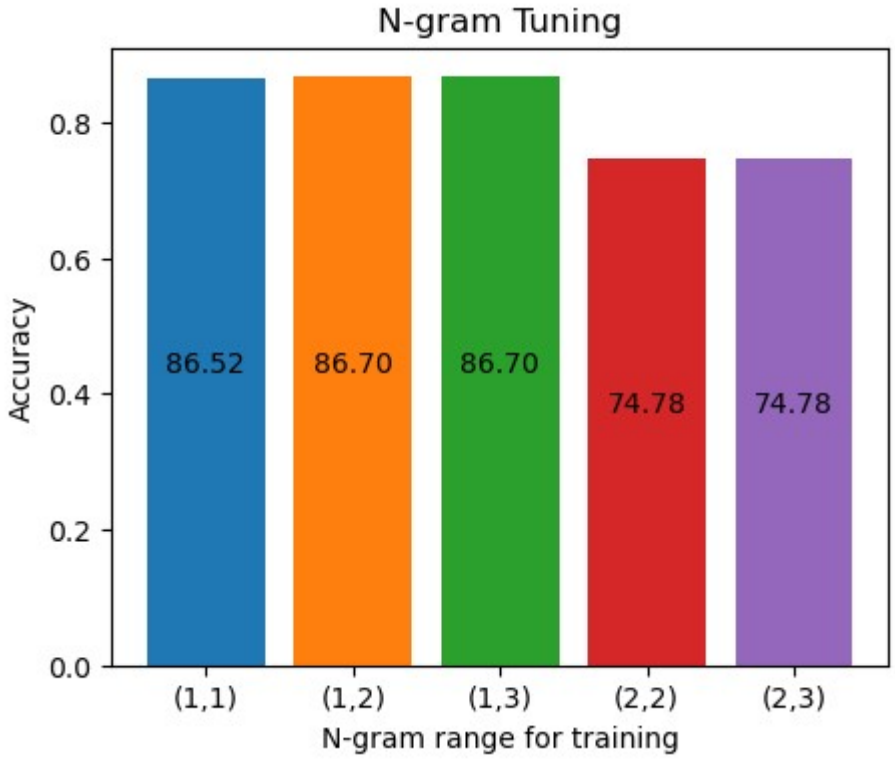
Out[ ]:
```
(571,)
```

The functions below were used for cross-validation. Stochastic Gradient Descent with a linear SVM loss function was used to classify in cross-validation. Any classifier could be used since performance within a classifier would be appropriate to test the difference in accuracy with different hyperparameters.

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Perceptron
from sklearn.svm import SVC

def FeatureExtraction(X_train, X_test,m,n,max_df=0.5,min_df=10):

    #Feature extraction
    vectorizer = TfidfVectorizer(strip_accents='unicode',stop_words='english',lower
    X_train_features = vectorizer.fit_transform(X_train)

    #Transforming validation set
    X_test_features = vectorizer.transform(X_test)

    return X_train_features, X_test_features, vectorizer.vocabulary_

def Classify(X_train_features,X_test_features,y_train,method='SGD_log_loss'):

    #Training classifier
    if method == 'SGD_log_loss':
        classifier = SGDClassifier(loss='log_loss',n_jobs=-1,random_state=seed)
    elif method == 'SGD_perceptron':
        classifier = SGDClassifier(loss='perceptron',n_jobs=-1,random_state=seed)
    elif method == 'Logistic':
        classifier = LogisticRegression(n_jobs=-1,random_state=seed)
    elif method == 'LinearSVM':
        classifier = SVC(random_state=seed,kernel='linear')
    elif method == 'Poly2SVM':
        classifier = SVC(random_state=seed,kernel='poly',degree=2)
    elif method == 'Poly3SVM':
        classifier = SVC(random_state=seed,kernel='poly',degree=3)
    elif method == 'Perceptron':
        classifier = Perceptron(n_jobs=-1,random_state=seed)
    else:
        raise Exception("Unsupported method passed")

    classifier.fit(X_train_features,y_train)

    #Predicting training set
    pred_train_labels = classifier.predict(X_train_features)

    #Predicting test set
    pred_test_labels = classifier.predict(X_test_features)

    return pred_train_labels, pred_test_labels
```

In [ ]:
```
#Stratified cross-validation for tuning of n-gram range
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score

kfold = StratifiedKFold(n_splits=5,shuffle=True,random_state=seed)

bestScore = 0
bestM = 0
bestN = 0

scores = []
params = []

for m in range(1,3):
    for n in range(m,4):
        avgTrain = 0
        avgVal = 0
        k = 1
        for train_i, test_i in kfold.split(X_train,y_train):

            #Getting feature matrices
            X_train_features, X_test_features, _ = FeatureExtraction(X_train[train_

            #Classification
            pred_train_labels, pred_test_labels = Classify(X_train_features,X_test_

            #Validity metric
            avgTrain += accuracy_score(y_train[train_i],pred_train_labels)
            avgVal += accuracy_score(y_train[test_i],pred_test_labels)

            k += 1

        avgTrain = avgTrain / kfold.get_n_splits()
        avgVal = avgVal / kfold.get_n_splits()

        scores.append(avgVal)
        params.append('(%d,%d)' % (m,n))

        #Store best
        if avgVal > bestScore:
            bestScore = avgVal
            bestM = m
            bestN = n
```

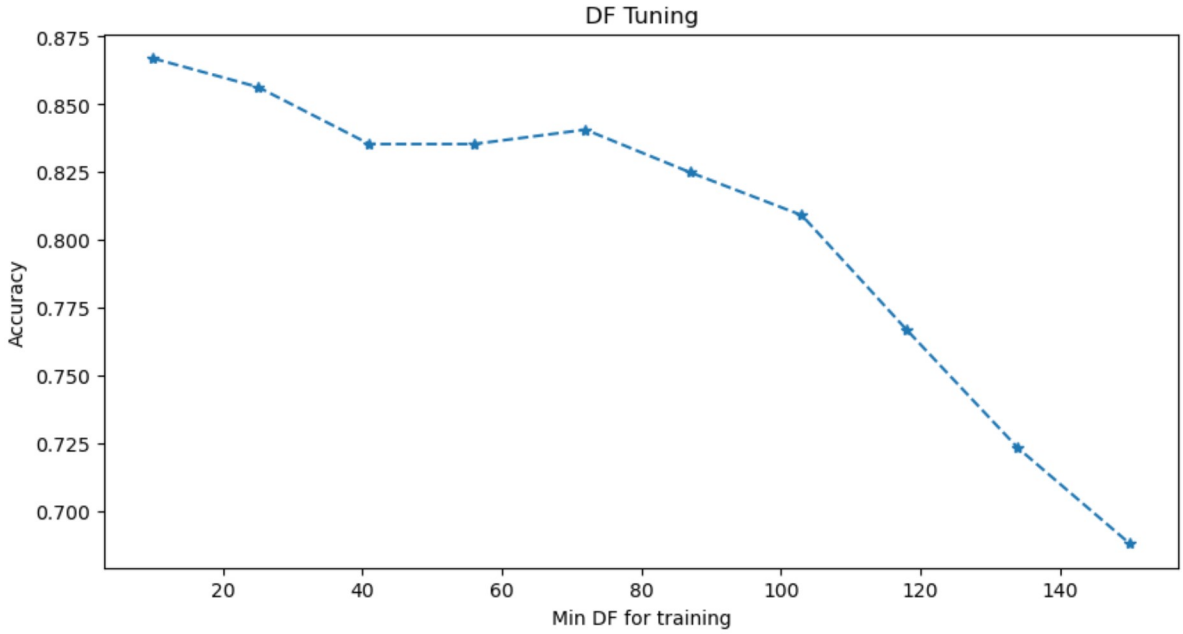
In [ ]:
```
#Plotting scores
figure = plt.figure(figsize=(5,4))
for i in range(len(params)):
    plt.bar(params[i],scores[i])
    plt.text(params[i],scores[i]/2,"%.2f" % (scores[i] * 100),ha="center")

plt.title("N-gram Tuning")
plt.xlabel("N-gram range for training")
plt.ylabel("Accuracy")
plt.show()
print("The best n-gram range is (%d,%d)" % (bestM,bestN), "with an average score of
```




```
The best n-gram range is (1,2) with an average score of 0.866957
```

In [ ]:
```
#Same as above but with min_df
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, confusion_matrix

kfold = StratifiedKFold(n_splits=5,shuffle=True,random_state=seed)

m = bestM
n = bestN
bestScore = 0
bestMinDf = 0
scores = []
params = []
mins = np.linspace(10,150,10,dtype=int)
max_df = 0.5

for min_df in mins:
    avgTrain = 0
    avgVal = 0
    k = 1
    for train_i, test_i in kfold.split(X_train,y_train):

        #Getting feature matrices
        X_train_features, X_test_features, _ = FeatureExtraction(X_train[train_i],X

        #Classification
        pred_train_labels, pred_test_labels = Classify(X_train_features,X_test_feat

        avgTrain += accuracy_score(y_train[train_i],pred_train_labels)
        avgVal += accuracy_score(y_train[test_i],pred_test_labels)

        k += 1

    avgTrain = avgTrain / kfold.get_n_splits()
    avgVal = avgVal / kfold.get_n_splits()
    scores.append(avgVal)
    params.append('(%d,%.2f)' % (min_df,max_df))

    if avgVal > bestScore:
        bestScore = avgVal
        bestMinDf = min_df
        bestMaxDf = max_df
```

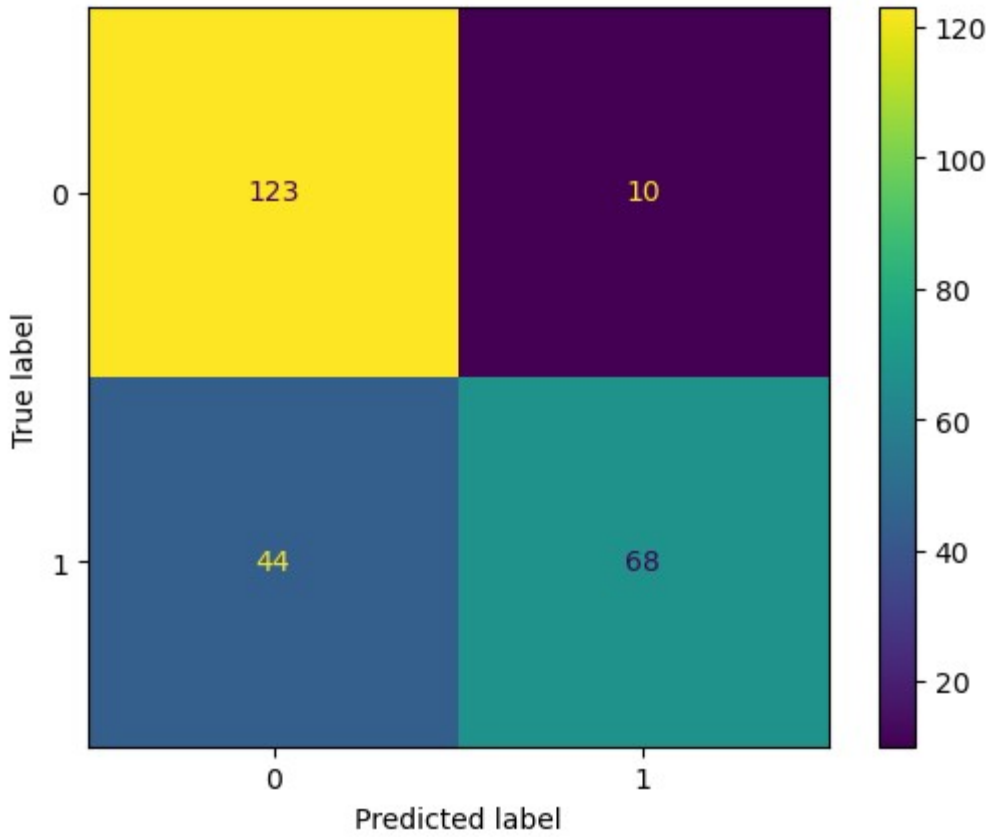
In [ ]:
```
#Plotting accuracy as a function of increasing min_df
figure = plt.figure(figsize=(10,5))

plt.plot(mins,scores,'--*')
plt.title("DF Tuning")
plt.xlabel("Min DF for training")
plt.ylabel("Accuracy")
plt.show()
```

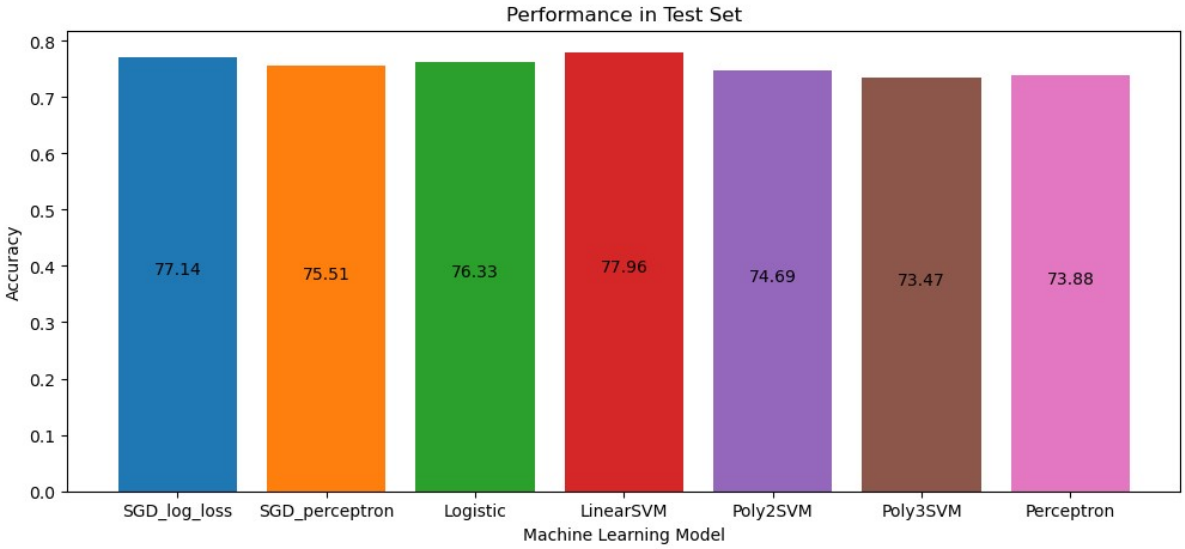


The best min and max df values are 10 and 0.50 respectively, with an average score of 0.866957

Training and Testing

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay

#Getting feature matrices
X_train_features, X_test_features, vocab = FeatureExtraction(X_train,X_test,2,3,max

#Classification
methods = ['SGD_log_loss','SGD_perceptron','Logistic','LinearSVM','Poly2SVM','Poly3
scores = []
for method in methods:
    pred_train_labels, pred_test_labels = Classify(X_train_features,X_test_features
    scores.append(accuracy_score(y_test,pred_test_labels))

    #Example confusion matrix, see report for clarification
    if method == 'LinearSVM':
        ConfusionMatrixDisplay.from_predictions(y_test,pred_test_labels)

#Plotting accuracy in test set
plt.figure(figsize=(12,5))
for i in range(len(methods)):
    plt.bar(methods[i],scores[i],0.8)
    plt.text(i,scores[i]/2,"%.2f" % (scores[i] * 100),ha="center")

plt.title("Performance in Test Set")
plt.xlabel("Machine Learning Model")
plt.ylabel("Accuracy")
plt.show()
```


123
10
44
68
0
1
True label
0
1
Predicted label
120
100
80
60
40
20


Performance in Test Set
Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
77.14
75.51
76.33
77.96
74.69
73.47
73.88
SGD_log_loss
SGD_perceptron
Logistic
LinearSVM
Poly2SVM
Poly3SVM
Perceptron
Machine Learning Model