wow much coin how money so crypto plz mine v rich very currency
wow
1 DOGECOIN

Overview

A hedgefund is seeking to profit from the volatility and inefficiences of the cryptocurrency market. They would like to focus on Dogecoin, a cryptocurrency with a $30B market cap. Their marketing team desires to better understand the sentiment around dogecoin, for understanding other owners and what words or phrases most resonnate with them.

Dogecoin began as satire, using a meme of a Shiba Inu dog misspelled as "doge." The makers, Billy Marcus and Jackson Palmer, created it in late 2013. They then launched a large campaign including sending "the Jamaican Bobsliegh team to the 2014 Olympics ... and sponsoring a Nascar driver" (Forbes). As a cryptocurrency, Dogecoin relies on block-chain cryptography. All coin holder's carry a ledger. And miner's have to solve math problems to create new chains; miners are rewarded for this work with dogecoin.

Dogecoin differs from many other cryptocurrencies in that it has no lifetime cap on the number of coins that can be produced. This means it is "highly inflationary" (Forbes), diminishing the coin as a store of value, but increasing its use as an actual currency.

Business Understanding

Market volatililty and the speculative nature of Dogecoin increase risk of owning the asset. Since the price of other cryptocurrencies have proven susceptible to the sentiment of its buyers and sellers (Frontiers in Physics), the hedfund seeks deeper comprehension of the sentiment of those very buyers and sellers, in order to get a better grasp on the market and possibly leverage that information for profits in the future. This project uses Twitter as a proxy for the buyers and sellers, and analyzes tweets containing the word "dogecoin" unto that end.

Data Understanding

This project used Twitter's Free API and Tweepy to scrape thousands of tweets. This came with some stipulations (that user information would not be shared) and some limitations. Tweets can only be scraped within the last 7 days, in increments of 180 tweets every 15 minutes, with a cap of 2,600 at one time. Thus, it takes 3 and 1/2 hours to scrape an accumulated 2,500 tweets.

Retweets were excluded because the Hedgefund is trying to get a sense of dogecoin audience and their sentiments. Retweets often were simply one bot retweeting any account that mentioned dogecoin. During one of our scrapings, bot retweets were almost 90% of the scraped tweets.

Twitter API Request Code

```
# Authentication
consumerKey = creds['API Key']
consumerSecret = creds['API Key Secret']
accessToken = creds['Access Token']
accessTokenSecret = creds['Access Token Secret']
auth = tweepy.OAuthHandler(consumerKey, consumerSecret)
auth.set_access_token(accessToken, accessTokenSecret)
api = tweepy.API(auth)

#Pull tweets based on Keyword and Amount (noOfTweet)
keyword = input('Please enter keyword or hastag to search: ')
noOfTweet = int(input ('Please enter how many tweets to analyze:
'))
tweets = tweepy.Cursor(api.search_tweets, q=keyword+' -
filter:retweets').items(noOfTweet)
tweet_list = []
tweet_date_list = []
for tweet in tweets:
    tweet_list.append(tweet.text)
    tweet_date_list.append(tweet.created_at)

tweet_list = pd.DataFrame(tweet_list)
```

Imports

In [ ]:

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

from textblob import TextBlob
```

```
import sys
import tweepy
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import os
import nltk
import pycountry
import re
import string
import json

# from wordcloud import WordCloud, STOPWORDS
# from PIL import Image
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.sentiment import SentimentAnalyzer
from langdetect import detect
from nltk.stem import SnowballStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score
```

Concatenate Tweets

Because Twitter's API has a limit, we had to scrape tweets in batches. Below, we concatenate all these serparate pulls, totaling to 4547 tweets.

```
In [ ]: df_1 = pd.DataFrame(pd.read_csv('data/doge_tweets_111621_1138',index_col=0)['tex
        df_2 = pd.DataFrame(pd.read_csv('data/dogecoin2_11_16_21_1pm.csv',index_col=0)['
        df_3 = pd.DataFrame(pd.read_csv('data/dogecoin_11_17_21_10am.csv', index_col=0)[
        df_4 = pd.DataFrame(pd.read_csv('data/doge_tweets_111721_1454.csv',index_col=0)[
        df_5 = pd.DataFrame(pd.read_csv('data/dogecoin_11_14_21_3pm.csv',index_col=0)['t
        df=pd.concat([df_1,df_2, df_3,df_4,df_5])
        df.drop_duplicates(inplace=True)
```

Write a function for vectorizing

The below functions build our vectorized data. Note, this function removes stopwords and sorts the vectorized. The function also returns a list of the top occurring words. These functions are used later on.

```
In [ ]: def get_top_n_gram(corpus,ngram_range,n=None):
            vec = CountVectorizer(ngram_range=ngram_range, stop_words = 'english',max_df
            bag_of_words = vec.transform(corpus)
            sum_words = bag_of_words.sum(axis=0)
            words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.ite
            words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
            return words_freq[:n], bag_of_words
```

```
In [ ]: def get_top_n_gram_test(corpus, trained_corpus, ngram_range,n=None):
            vec = CountVectorizer(ngram_range=ngram_range, stop_words = 'english',max_df
            bag_of_words = vec.transform(corpus)
            sum_words = bag_of_words.sum(axis=0)
            words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.ite
            words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
            return words_freq[:n], bag_of_words
```

Clean Tweets

Firstly, We removed punctuation and lowercase each word, in order to track similar words. Likewise, we made every word lower case.

In [ ]:
```
# Remove RT
remove_rt = lambda x: re.sub('RT @\w+: '," ",x)
# Remove punctuation
rt = lambda x: re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+://\S+)"," ",x)
# Remove Retweets
df["clean_text"] = df.text.map(remove_rt).map(rt)
# Lower the case
df["clean_text"] = df.clean_text.str.lower()
```

Using Sentiment Intensity Analyzer with VADER to Build Our Target Variable

The below cell uses textblob and sentiment intensity analyzer and creates a dataframe with the polarity, subjectivity and sentiment of the tweet.

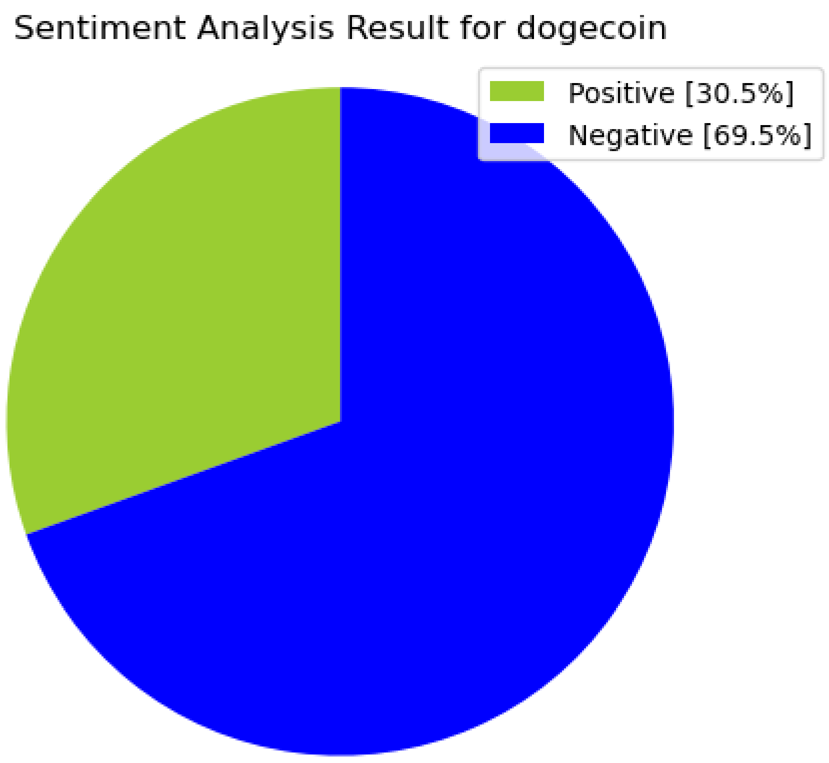
In [ ]:
```
#Calculating Negative, Positive, Neutral and Compound values
df_probs = pd.DataFrame()
df_probs[['polarity', 'subjectivity']] = df['clean_text'].apply(lambda Text: pd.
for index, row in df['text'].iteritems():
    score = SentimentIntensityAnalyzer().polarity_scores(row)
    neg = score['neg']
    neu = score['neu']
    pos = score['pos']
    comp = score['compound']
    if neg > pos:
        df_probs.loc[index, 'sentiment'] = 0
    elif pos > neg:
        df_probs.loc[index, 'sentiment'] = 1
    else:
        df_probs.loc[index, 'sentiment'] = 0
df_probs.head(10)
```

Out[ ]:

	polarity	subjectivity	sentiment
0	0.000000	0.000000	0.0
1	0.000000	0.000000	0.0
2	0.000000	0.350000	1.0
3	0.000000	0.000000	1.0
4	0.000000	0.000000	1.0
5	0.000000	0.000000	0.0
6	0.309028	0.729167	1.0
7	0.445455	0.351515	1.0
8	-0.400000	0.400000	0.0
9	0.000000	0.000000	0.0

Here is the piechart of the breakdown of tweets after the target column is created.

```
In [ ]: def percentage(part,whole):
         return 100 * float(part)/float(whole)

        positive = format(percentage(df_probs[df_probs['sentiment'] == 1].shape[0],df_pr
        negative = format(percentage(df_probs[df_probs['sentiment'] == 0].shape[0],df_pr

        labels = ['Positive ['+str(positive)+'%]','Negative ['+str(negative)+'%]']

        sizes = [positive, negative]
        colors = ['yellowgreen', 'blue']
        patches, texts = plt.pie(sizes,colors=colors, startangle=90)
        plt.style.use('default')
        plt.legend(labels)
        plt.title("Sentiment Analysis Result for dogecoin" )
        plt.axis('equal')
        plt.show()
```



Feature Engineering our Positive Reinforcement Variable

We filter for only positive tweets as marked by the sentiment intensity analyzer and make a dataframe.

```
In [ ]: df.reset_index(drop=True, inplace=True)
        df_pos = df[['text']].copy(deep=True)
        df_probs.reset_index(drop=True, inplace=True)
        df_pos['sentiment'] = df_probs['sentiment'].copy(deep=True)
        df_pos = df_pos[df_pos['sentiment'] == 1]
```

Tweets are cleaned of punctuation and capitalization.

```
In [ ]: # Remove RT
        remove_rt = lambda x: re.sub('RT @\w+: '," ",x)
        # Remove punctuation
        rt = lambda x: re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+://\S+)"," ",x)
        # Remove Retweets
        df_pos["clean_text"] = df_pos.text.map(remove_rt).map(rt).copy(deep=True)
        # Lower the case
        df_pos["clean_text"] = df_pos.clean_text.str.lower().copy(deep=True)
```

Finally we run our n_gram finder function to generate a list of the top 300 frequently occurring words that exist in only positive tweets.

```
In [ ]: positives, pos_vec = get_top_n_gram(df_pos['clean_text'], (1,1), 300)
        positives = [X[0] for X in positives]
        positives = [x for x in positives if x not in ['dogecoin','doge','twitter','soci
        print(positives)
```

```
['new', 'crypto', 'earn', 'friend', 'current', 'alert', 'whale', 'price', 'bitco
in', 'pasive', 'income', 'day', 'like', 'update', 'btc', 'support', 'shiba', 'go
od', 'value', 'shib', 'just', 'worth', 'says', 'eth', 'swap', 'buy', 'usd', 'ad
d', 'ethereum', 'ba', 'amp', 'elon', 'inu', 'minute', 'legacy', 'token', 'join',
'change', 'great', 'coins', 'best', 'nft', 'mdoge', 'love', 'empire', 'coin', 'c
ryptocurrency', 'free', 'miss', 'moon', 'future', 'want', 'community', 'huge',
'check', 'shibainu', 'important', 'addr', 'better', 'cash', 'time', 'game', 'hop
e', 'awesome', 'accept', 'ready', 'team', 'make', 'amazing', 'win', 'let', 'soo
n', 'big', 'tesla', 'smart', 'dogearmy', 'looking', 'going', 'social', 'bnb', 'l
ol', 'strong', 'litecoin', 'nice', 'happy', 'know', 'memecoin', 'floki', 'mone
y', 'bwcdeals', 'yes', 'market', 'ad', 'solana', 'help', 'binance', 'thanks', 'm
usk', '2021', 'defi', '100', 'wow', 'dog', 'run', 'follow', 'coming', 'ada', 'wo
rld', 'tippingtuesday', 'devs']
```

Now the column is created by checking for the existence of each word in the positive list we just created against each tweet. If a positive word exists in the tweet its positive word counter goes up by one.

```
In [ ]: from nltk import sent_tokenize
        from nltk import word_tokenize

        def extract_features(text, top_100):
            sia = SentimentIntensityAnalyzer()
            features = dict()
            wordcount = 0
            compound_scores = list()
            positive_scores = list()

            for sentence in sent_tokenize(text):
                for word in word_tokenize(sentence):
                    if word.lower() in top_100:
                        wordcount += 1
                compound_scores.append(sia.polarity_scores(sentence)["compound"])
                positive_scores.append(sia.polarity_scores(sentence)["pos"])

            # Adding 1 to the final compound score to always have positive numbers
            # since some classifiers you'll use later don't work with negative numbers.
            features["wordcount"] = wordcount

            return features['wordcount']

        features = [
            (extract_features(review, positives))
            for review in df['clean_text']]
```

Adding the column to the main dataframe

```
In [ ]: df_feat = pd.concat([df['clean_text'], pd.DataFrame(features,columns=['positive_
```

Train, Test, Split of Data

The below cell shows our two features of the tweet text and the count of positive words in each tweet.

```
In [ ]: # Train-test split
        X_train,X_test,y_train,y_test = train_test_split(df_feat, df_probs['sentiment'],
        X_train
```

Out[ ]:

	clean_text	positive_word_count
2253	businessgrowth tumblr twitter facebook in...	0
1310	what do you do when the crypto takes a huge di...	2
2226	btfd gt buy the f king dogecoin	1
1228	businesswoman tumblr twitter facebook ins...	0
579	business businessgoals businessgrowth busi...	1
...	...	...
905	not so mr wonderful says dogecoin holders di...	2
5192	this clean energy manufacturing company has a ...	3
3980	empire there s some things you can t buy wit...	2
235	saitama shiba i ll have all the ca...	1
5157	tweet del d a	0

4547 rows × 2 columns

The tweet text is then vectorized and the positive word counts are concatenated back to the vectorized data.

```
In [ ]: feat, feat_vec_train = get_top_n_gram(X_train['clean_text'], (1,1), 100)
        feat, feat_vec_test = get_top_n_gram_test(X_test['clean_text'], X_train['clean_t

        X_train.reset_index(drop=True, inplace=True)
        X_test.reset_index(drop=True, inplace=True)

        X_train_vec = pd.concat([X_train['positive_word_count'], pd.DataFrame(feat_vec_t
        X_test_vec = pd.concat([X_test['positive_word_count'], pd.DataFrame(feat_vec_tes
```

Baseline

Our baseline is the average of the majority class, which is 69% neutral/negative.

```
In [ ]: y_train.value_counts()[0]/y_train.shape[0]
```

```
Out[ ]: 0.6927644600835716
```

The Model

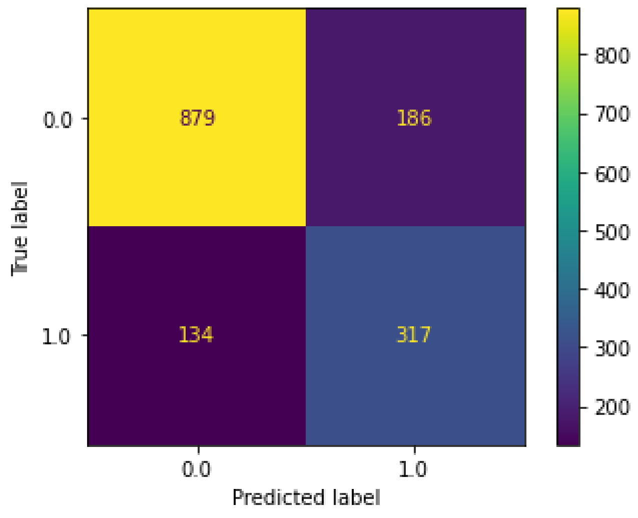
For our final model we use multinomial naive bayes and feed in our vectorized data and positive word counts. We use naive bayes to help prevent overfitting. The output of this cell shows the cross validation results, the test predictions and the accuracy score of predicting on test data. Our accuracy score was 78% on the test data.

```
In [ ]:
# Naive Bayes
nb = MultinomialNB()
nb.fit(X_train_vec,y_train)

#Cross-Validation
from sklearn.model_selection import cross_val_score
print(cross_val_score(nb, X_train_vec, y_train))

# Predictions
y_preds = nb.predict(X_test_vec)
df_fin = pd.DataFrame()
df_fin['predictions'] = y_preds
print(df_fin.value_counts('predictions'))

# Metric on Test Score
print('Accuracy Score:',accuracy_score(y_test,df_fin['predictions']))
```

```
[0.78901099 0.7967033  0.78767877 0.7909791  0.75137514]
predictions
0.0    1013
1.0     503
dtype: int64
Accuracy Score: 0.7889182058047494
```

Confusion Matrix

```
In [ ]:
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(nb, X_test_vec, y_test)
plt.show()
```



Recommendations

This project revealed that most of the confidence in dogecoin is built around hype. For this reason we would want to use our model to continue to monitor dogecoin sentiment to gauge our own confidence in the coin.

It's also clear that a lot of the language used for dogecoin is slang and not always intuitive. So we want to track, understand and then correctly use the buzz words in our social media campaigns. This will strengthen outreach towards investors interested in looking at dogecoin.

Future Research

For this project we only tracked single/double words and tied them to the positive class. For future research we'd like to look at words for the negative/neutral class and also incorporating phrases to better improve the model performance.

Finally we'd like to try and apply our sentiment analysis towards predicting the future price of dogecoin. We would look into seeing if there is a correlation between social media sentiment around the coin and how the coin price changes.