Introdução

Bitcoins e outras criptomoedas estão sendo muito exploradas no mundo atualmente e sua variação tem um valor elevado.

Juntamente com o mundo conectado que vivemos, podemos acompanhar em redes sócias informações on-line sobre bitcoins, informações essas que nos ajuda tomar decisões se está na hora de entrar nesse mercado ou não.

Por esse motivo, me senti motivado a minerar o Twitter em busca uma ajuda para identificar se o que estão falando no momento sobre bitcoin é bom ou ruim e possivelmente utilizar essa informação para acompanhar o mercado de Bitcoins para poder tomar uma decisão de quando será o melhor momento para a compra dessa criptomoeda.

Definição do Problema

Primeiramente capturei em dias diferentes uma quantidade razoável de Tweets, o qual são textos publicados no Twitter, que contenham a palavra bitcoin. Após a captura desses Tweets, analisei e classifiquei eles manualmente como Positivo, Negativo ou Neutro. Após essa classificação usei esses textos (Tweets) como entrada para um modelo de classificação.

Tweets com a palavra Bitcoin Classificados previamente

Treinamento e Teste

Tweets Novos

Previsão

Correlação entre a classificação prevista e o valor do Bitcoin

Conjunto de dados

Para realização desse projeto usei os dados coletados do Twitter, no caso aproximadamente 270.000 Tweets.

As informações coletadas foram:

- **USR**
  - Nome do usuário do Twitter que publicou esse Tweet
- **TWEET**
  - O texto publicado pelo usuário, o qual sempre terá a palavra bitcoin.
- **LANG**
  - Linguagem em que o Tweet foi publicado
- **DATE**
  - Data de publicação do Tweet
- **CLAS**
  - No ato da captura dos tweets todos vieram como Neutro, mas em seguida eu classifiquei alguns como veremos na seção seguinte, sendo: Positiva, para as publicações que me influenciariam a comprar bitcoin: Negativa, para as publicações que me influenciariam a não comprar bitcoin: Neutro, para as publicações que não tem influência na compra de bitcoin.

Para captura dos dados do Twitter utilizei um código em Python criado por mim, mas usando como referência:

https://apps.twitter.com/app/14711331/keys
https://ronanlopes.me/coletor-de-tweets-em-python-com-o-tweepy/

In [2]:
```
#https://apps.twitter.com/app/14711331/keys
#https://ronanlopes.me/coletor-de-tweets-em-python-com-o-tweepy/
import tweepy
import pandas as pd
import datetime
```

In [2]:
```
#Autenticações
consumer_key = '##############################'
consumer_secret = '##############################'
access_token = '##############################'
access_token_secret = '##############################'

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)
```

In [3]:
```
res = pd.DataFrame(columns=['USR', 'TWEET', 'LANG', 'DATE', 'CLAS'])
```

In [4]:
```
#Coletando tweets
class CustomStreamListener(tweepy.StreamListener):

  def on_status(self, tweet):
    try:
        res.loc[len(res)] = [str(unicode(tweet.author.screen_name).encode("utf-8")),
                             str(unicode(tweet.text).encode("utf-8")),
                             str(unicode(tweet.lang).encode("utf-8")),
                             str(unicode(tweet.created_at).encode("utf-8")),
                            "NEUTRO"]
        #print tweet
        if len(res.loc[res['LANG']=='pt']) < 201:
            if str(unicode(tweet.lang).encode("utf-8"))=='pt':
                print len(res.loc[res['LANG']=='pt'])
            return True
        else: return False

    except:
        return True

  def on_error(self, status_code):
      print "Erro com o código:", status_code
      return True

  def on_timeout(self):
      print "Tempo esgotado!"
      return True
```

```
In [5]: # Pegando a consulta por parâmetro
consulta = [u'bitcoin']

#Criando o coletor com timeout de 60 seg
try:
    streaming_api = tweepy.streaming.Stream(auth, CustomStreamListener(), timeout=60)
    streaming_api.filter(follow=None, track=consulta, languages=["pt", "en"])

except:
    print "ERRO"
```

Exploração de dados

Com do tweets adquiridos passei para a analise do conteúdo. Conforme imagem abaixo, menos de 1% dos tweets eram do idioma português, então decidi abandar os tweets em português e manter apenas os que estavam na língua inglesa.

```
In [5]: data.count()
Out[5]: USR      271859
        TWEET    271859
        LANG     271859
        DATE     271859
        CLAS     271859
        dtype: int64

In [6]: data['LANG'].count()
Out[6]: 271859

In [7]: data.loc[data['LANG']=='en'].count()
Out[7]: USR      269810
        TWEET    269810
        LANG     269810
        DATE     269810
        CLAS     269810
        dtype: int64

In [8]: data.loc[data['LANG']=='pt'].count()
Out[8]: USR      2049
        TWEET    2049
        LANG     2049
        DATE     2049
        CLAS     2049
        dtype: int64
```

Restando apenas os tweets em inglês passei para a fase de classificação manual dos tweets, classificando-os como Positivo, Negativo e Neutro conforme visto na seção anterior. Entretanto, classificar aproximadamente 200.000 tweets não era viável, então ordenei os tweets de forma aleatória, abandonando assim a data em que o tweet foi postado. Nessa limpeza já removi o usuário que escreveu o tweet, pois para esse projeto o usuário não era necessário. Feito isso fiquei com uma base com apenas tweets em inglês contendo apenas as colunas TWEET, LANG (que não foi utilizada) e CLAS, como podem ser vistas na imagem abaixo:

```
In [5]: ree-Machine_Learning\Trabalhos\Projetos\ProjetoFinal\Saidas\saida_final.csv', sep=';', encoding='utf-8')
```

Out[5]:

	USR	TWEET	LANG	DATE	CLAS
0	M74OHkZlGsqVLRv	@DuongTien990 Bitcoin Gold (BTG) successfully ...	en	09/02/2018 13:51	NEUTRO
1	whats_a_bitcoin	If you want to earn money by mining bitcoins, ...	en	09/02/2018 13:51	NEUTRO
2	Openthemag	At its peak, Bitcoin was worth over a quarter ...	en	09/02/2018 13:51	NEUTRO
3	CoinsAnalyst	🔥 ICO PRE-SALE: 35% BONUS 🔥\r\n\r\nTOKIA: th...	en	09/02/2018 13:51	NEUTRO
4	Sanjay_Uppal	@M1Singapore .... I keep receiving spam calls ...	en	09/02/2018 13:51	NEUTRO

```
In [6]: degree-Machine_Learning\Trabalhos\Projetos\ProjetoFinal\Saidas\Tratadas.csv", sep=';', encoding='utf-8')
```

Out[6]:

	TWEET	LANG	CLAS
0	RT @pyramuscrypto: Out here trying to find the...	en	POSITIVO
1	Let's farm bitcoins for free together - https:...	en	NEUTRO
2	The Japanese government said today that inspec...	en	NEGATIVO
3	With this project we will help undervalued coi...	en	POSITIVO
4	Op-ed: How Decentralized Protocols Are Threate...	en	NEUTRO

Após essa alteração no arquivo dos tweets iniciei minha classificação manual seguindo os critérios preestabelecidos, porém após mais de 15.000 tweets classificados identifiquei que a maioria era neutro, tendo uma proporção de 0,6% Negativos, 7% Positivos e 92,4% Neutros. Então decidi continuar a classificação buscando apenas os tweets Negativos e Positivos para chegar em um dataset equilibrado entre os Positivos, Negativos e Neutros.

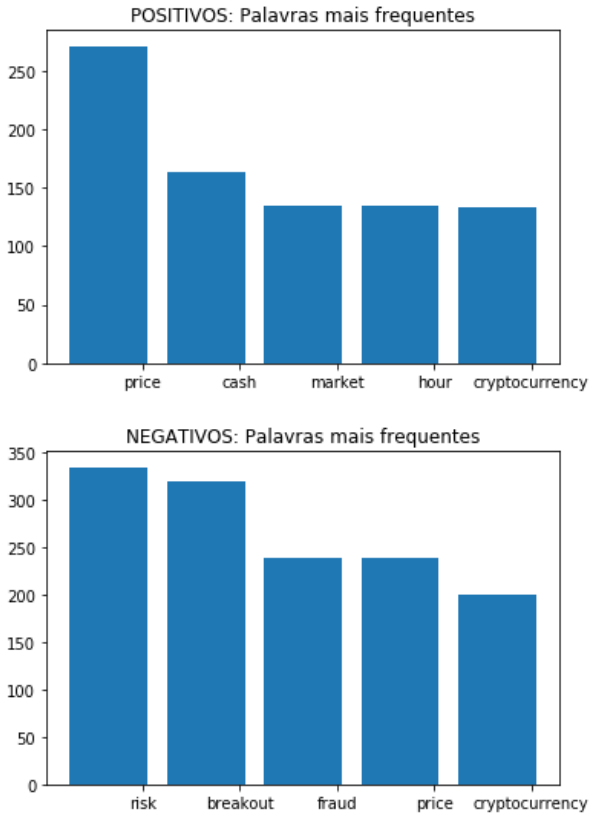
Feito isso cheguei em novo dataset com 5.000 tweets classificados conforme abaixo:

```
In [12]: data.CLAS.value_counts()

Out[12]: NEUTRO      1710
         POSITIVO    1666
         NEGATIVO    1624
         Name: CLAS, dtype: int64

In [ ]:
```

Para efeito exploratório, inclui um histograma de frequência, abaixo, para identificar quais eram as 5 palavras mais utilizadas para os tweets classificados como POSITIVO e as 5 palavras mais utilizadas para os tweets classificados como NEGATIVO.



Dessa forma tenho um dataset balanceado para poder iniciar o processo de criação de uma modelo, porém, antes do inicio da modelagem identifiquei que muitos tweets possuíam palavras que não deveriam ser analisadas, como “RT” (retweet),

“emoji”, “@” (antecede o nome de um usuário). Para esses casos eu criei o código abaixo para fazer uma limpeza do texto dos tweets.

In [6]:
```
##CRIAÇÃO DE DATAFRAME VAZIO PARA TRATAMENTO DOS DADOS
X_all_DF = pd.DataFrame(columns=['TWEET'])

for linha in X_raw:
    words = word_tokenize(linha) ##SEPARA POR PALAVRAS
    novalinha = ""
    flag = True
    for w in words:
        if w != "RT": ##NÃO UTILIZAR A PALAVRA RT, RETWEET
            if re.match('[a-zA-Z0-9_]',w) and flag: ##UTILIZAR APENAS PALAVRAS ALFANUMERICAS
                novalinha+=w+" " ##RECONSTREI A FRASE SEM RT, SEM @, SEM PONTUAÇÕES
            if w == "@": ##VERIFICAR SE É UM @, SE FOR MARCA PARA NÃO UTILIZAR A PROXIMA PALAVRA,
                            ##POIS TRATA-SE DE SITAÇÃO DE USUARIO
                flag = False
            else:
                flag = True
    X_all_DF.loc[len(X_all_DF)] = novalinha ## INCLUI LINHA NOVA, TRATADA, NO DATAFRAME NOVO

X_all = X_all_DF['TWEET'] ##CRIA NOVO DATAFRAME PARA SER UTILIZADO COMO FEATURES
```

Algoritmos e Técnicas

Pela natureza desse projeto, onde gostaria de identificar o sentimento de cada tweet mencionando a palavra bitcoin, algumas técnicas recomendadas são Bag of Words, remoção de Stop Words, Stemming, criação de um Corpus e Word2Vec.

Decidi utilizar a criação de bag of words, onde conteve todas as palavras da minha base de treino vetorizadas para analise de quantidade de palavras em cada tweet. Além de considerar o set de stopwords da biblioteca NLTK (https://pythonspot.com/nltk-stop-words/) para filtrar palavras que não possuem peso para a analise de texto, como: the, is, are....ente outras palavras da língua inglesa.

Além de considerar stop words, decidi utilizar a técnica de Stemming, também da biblioteca NLTK (http://www.nltk.org/howto/stem.html), para reduzir as palavras para seu radical. Essa técnica permite trabalhar com palavras que possuem o mesmo radical como se fossem a mesma palavra, dessa forma podemos dar mais peso para as palavras, pois estamos considerando todas as formas que essa palavra possa ter aparecido como apenas uma forma.

Para vetorização, remoção de stop words e stemming das palavras eu criei o código abaixo:

In [6]:
```
stopWords = set(stopwords.words('english')) ## STOPWORDS DA BIBLIOTECA NLTK
ps = PorterStemmer() ##STEMMING DA BIBLIOTECA NLTK
vectorizer = CountVectorizer() ##INSTANCIA PARA VETORIZAR E CONTAR AS PALAVRAS EXISTENTES NO TWEET

X_all_bag_DF = pd.DataFrame(columns=['TWEET']) ## NOVO DATAFRAME PARA UTILIZAÇÃO DE BAG OF WORDS

for linha in X_all:
    words = word_tokenize(linha) ##SEPARA POR PALAVRAS
    novalinha = ""
    for w in words:
        if w not in stopWords: ##VERIFICAR SE ESTA NA LISTA DE STOPWORDS, SE NÃO ESTIVER CONTINUA
            novalinha+=ps.stem(w)+" " ##INCLUI O RADICAL DA PALAVRA EM UMA NOVA LINHA
    X_all_bag_DF.loc[len(X_all_bag_DF)] = novalinha ##INCLUI LINHA NO NOVO DATAFRAME

X_all_bag = X_all_bag_DF['TWEET'] ##CRIA NOVO DATAFRAME PARA SER UTILIZADO COMO FEATURES,
                                  ###JÁ EXCLUINDO STOPWORDS E REDUZINDO AS PALAVRAS PARA SEU RADICAL
```

In [7]:
```
##VETORIZAR E CONTAR AS PALAVRAS EXISTENTES NO TWEET E ARMAZENAR NA VARIAVEL X
X = vectorizer.fit_transform(X_all_bag).toarray()
```

Com o dataframe pronto, vetorizado, utilizei um holdout com 75% da base para treino e os outros 25% para teste. Separados dessa forma apliquei 3 algoritmos que são indicados para analise de texto, 2 tipos diferentes de Naive Bayes pois ele trabalha bem com analise de texto exatamente por considerar que cada palavra é um componente independente, além de um Random Forest, pois assim como arvore de decisão o RF é um ótimo classificador.

Para entender melhor a diferença entre os modelos escolhido, seguem algumas referências:

NAIVE BAYES

(ref: https://www.vooo.pro/insights/6-passos-faceis-para-aprender-o-algoritmo-naive-bayes-com-o-codigo-em-python/

Ref2: https://stats.stackexchange.com/questions/33185/difference-between-naive-bayes-multinomial-naive-bayes?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa).

- **APLICAÇÃO REAL**: Classificação de textos/Filtragem de spam/Análise de sentimento: classificadores Naive Bayes utilizados principalmente em classificação de textos (devido a um melhor resultado em problemas de classes múltiplas e regra de independência) têm maior taxa de sucesso em comparação com outros algoritmos. Como resultado, é amplamente utilizado na filtragem de spam (identificar spam) e Análise de Sentimento (em análise de mídia social, para identificar sentimentos positivos e negativos dos clientes).
- **PROS**: Quando a suposição de independência prevalece, um classificador Naive Bayes tem melhor desempenho em comparação com outros modelos como regressão logística, e você precisa de menos dados de treinamento. O desempenho é bom em caso de variáveis categóricas de entrada comparada com a variáveis numéricas. Para variáveis numéricas, assume-se a distribuição normal (curva de sino, que é uma suposição forte).
- **CONTRAS**: Uma limitação do Naive Bayes é a suposição de preditores independentes. Na vida real, é quase impossível que ter um conjunto de indicadores que sejam completamente independentes. Se a variável

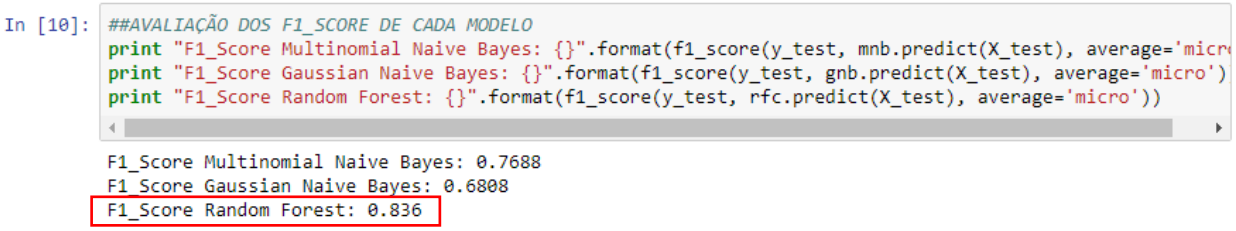
categórica tem uma categoria (no conjunto de dados de teste) que não foi observada no conjunto de dados de treinamento, então o modelo irá atribuir uma probabilidade de 0 (zero) e não será capaz de fazer uma previsão. Isso é muitas vezes conhecido como "Zero Frequency". Para resolver isso, podemos usar a técnica de alisamento. Uma das técnicas mais simples de alisamento é a chamada estimativa de Laplace.

- **Diferença em Gausian NB e Multnomial NB**: O termo Multinomial Naive Bayes simplesmente nos permite saber que cada é uma distribuição multinomial, ao invés de alguma outra distribuição. Isso funciona bem para dados que podem ser facilmente convertidos em contagens, como contagens de palavras no texto, o que foi feito nesse projeto.

RANDOM FOREST
(ref: https://dimensionless.in/introduction-to-random-forest/)

- **APLICAÇÃO REAL**: Random Forest são como varias arvores de decisão trabalhando juntas para atingirem o melhor resultado. A escolha do RF para esse projeto vem principalmente por um dos pontos fortes dele, que é ser considerado um dos melhores modelos de decisão existentes. Várias áreas utilizam RF, como: reconhecimento de objetos, analise de sequência de aminoácidos e classificação de galáxias em astronomia.
- **PROS**: É um dos modelos de decisão mais preciso, funciona bem em grandes conjuntos de dados, pode ser usado para extrair as variáveis mais importantes e não requer feature engineering (scaling e normalização).
- **CONTRAS**: Pode dar overfitting em casos de dados ruidosos, ao contrario das arvores de decisão os resultados do RF são difíceis de interpretar e os parâmetros (hiperparametros) precisão de um bom ajuste para ter uma boa precisão, por isso utilizamos com Grid Search

Para analisar qual dos modelos teve um melhor desempenho utilizei o F1_Score, pois assim consigo mensurar a precisão e o recall do modelo em um indicador único. Tendo como resultado para o melhor modelo o Random Forest, conforme abaixo:



Dado o melhor modelo, considerando o F1_Score dos dados de teste, decidi tentar melhorar o modelo buscando um melhor F1_Score atrás do Grid Search, o qual executa o modelo alterando os parâmetros previamente definidos buscando quais são os parâmetros necessários para se alcançar o melhor modelo.

Para o Random Forest escolhi os parâmetros "n_estimators" (numero de arvores na floresta), "criterion" (forma e qualidade de como será dividido os dados nas arvores), "max_depth" (profundidade máxima de cada arvore), "min_samples_split" (número

mínimo de amostras em cada separação), “min_samples_leaf” (número mínimo de amostrar para compor uma folha da arvore). Abaixo segue codificação:

Obtive os seguintes resultados:

In [16]:
```
##VERIFICAR QUAIS FORAM OS MELHORES VALORES DOS PARAMETROS
print "O parâmetro 'n_estimators' é {} para o modelo ótimo.".format(clf.get_params()['n_estimators'])
print "O parâmetro 'max_depth' é {} para o modelo ótimo.".format(clf.get_params()['max_depth'])
print "O parâmetro 'min_samples_leaf' é {} para o modelo ótimo.".format(clf.get_params()['min_samples_l
print "O parâmetro 'criterion' é {} para o modelo ótimo.".format(clf.get_params()['criterion'])
print "O parâmetro 'min_samples_split' é {} para o modelo ótimo.".format(clf.get_params()['min_samples_

print " "

##VERIFICAR O F1_SCORE DO MODELO CALIBRADO COM O BEST ESTIMATOR DO GRIDSEARCH
print "O modelo calibrado tem F1 de {} no conjunto de treinamento.".format(f1_score(y_train, clf.predic
print "O modelo calibrado tem F1 de {} no conjunto de teste.".format(f1_score(y_test, clf.predict(X_tes
```

```
O parâmetro 'n_estimators' é 100 para o modelo ótimo.
O parâmetro 'max_depth' é 100 para o modelo ótimo.
O parâmetro 'min_samples_leaf' é 1 para o modelo ótimo.
O parâmetro 'criterion' é gini para o modelo ótimo.
O parâmetro 'min_samples_split' é 5 para o modelo ótimo.

O modelo calibrado tem F1 de 0.9368 no conjunto de treinamento.
O modelo calibrado tem F1 de 0.8584 no conjunto de teste.
```

Com esses ajustes obtidos no GridSearch temos um modelo com F1_Score 2% melhor que o modelo anterior. O ganho não foi muito significante, mas como não houve um aumento considerável de tempo de execução considero que esse modelo seja melhor que o anterior. Podemos notar também que o modelo está fit, pois tanto o treino quando o teste estão com F1_Score próximos e altos, mostrando que o modelo acerta bem para os dados novos e para os dados de treino.

Modelo de referência

Após o desenvolvimento desse modelo um modelo de benchmark foi criado para aferir melhor a qualidade do meu modelo. Como benchmark para validar a sanidade do meu modelo final, utilizei uma classificação aleatória, o DummyClassifier do sklearn (http://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html), afim de medir se meu modelo tem resultados melhores ou piores que a classificação aleatória.

In [12]:
```
##MELHOR F1_SCORE ACIMA = RANDOM FOREST
clf = RandomForestClassifier() #NOVA INSTANCIA DE RANDOM FOREST

##DEFINIÇÃO DOS PARAMENTROS A SEREM TESTADOS NO RANDOM FOREST
parameters = {'n_estimators' : [1, 10, 100],
              'criterion' : ['gini', 'entropy'],
              'max_depth' : [1, 10, 100],
              'min_samples_split' : [2, 5, 10],
              'min_samples_leaf' : [1, 5, 10]}

grid_obj = GridSearchCV(clf, parameters, cv=10)##CRIA OBJETO DE GRIDSEARCH COM O CLASSIFICADOR E
                                               ##PARAMETROS ACIMA E CROSSVALIDATION DE 10 FOLDS
grid_obj = grid_obj.fit(X_train, y_train)##RODAR O GRIDSEARCH PARA IDENTIFICAR OS MELHORES PARAMETROS E
                                         ##TREINAR O MODELO COM ELES

clf = grid_obj.best_estimator_ ##CRIA NOVO CLASSIFICADOR COM O MELHOR RESULTADO DO GRIDSEARCH ACIMA
```

Como podemos ver abaixo, a implementação do modelo benchmark aleatório, os resultados do F1_Score são bem piores que meu modelo, garantindo assim mais uma vez de que a classificação feita pelo meu modelo está boa.

```
In [13]: ##BENCHMARK CRIADO COM O DUMMY CLASSIFIER, QUE CLASSIFICA DE FORMA "BURRA OS TWEETS"
         dmc = DummyClassifier()

In [14]: dmc.fit(X_train, y_train) ##TREINA O BENCKMARK COM OS MESMOS DADOS PASSADOS PARA O MODELO
                                   ##TREINADO E OTIMIZADO ACIMA

Out[14]: DummyClassifier(constant=None, random_state=None, strategy='stratified')

In [15]: ##VERIFICAR O F1_SCORE DO MODELO BENCHMARK
         print "O modelo benchmark tem F1 de {} no conjunto de treinamento.".format(f1_score(y_train, dmc.predic
         print "O modelo benchmark tem F1 de {} no conjunto de teste.".format(f1_score(y_test, dmc.predict(X_tes

         O modelo benchmark tem F1 de 0.3336 no conjunto de treinamento.
         O modelo benchmark tem F1 de 0.3376 no conjunto de teste.
```

Conclusão

O desenvolvimento desse projeto me fez entender melhor como criar uma analise de sentimentos, mas mesmo tendo bons resultados eu ainda não utilizaria esse modelo no mundo real, pois o mercado de bitcoins é muito volátil. Gostaria de aperfeiçoar esse modelo em um momento futuro aplicando word2vec e outras formas mais sofisticadas de analise de sentimento como redes neurais.