Data Analysis & XGBoost Starter (0.35460 LB)

Python notebook using data from Quora Question Pairs · 112,780 views · 3y ago · data visualization, eda, nlp, +2 more

Identifying Duplicate Questions

Welcome to the Quora Question Pairs competition! Here, our goal is to identify which questions asked on Quora, a quasi-forum website with over 100 million visitors a month, are duplicates of questions that have already been asked. This could be useful, for example, to instantly provide answers to questions that have already been answered. We are tasked with predicting whether a pair of questions are duplicates or not, and submitting a binary prediction against the logloss metric.

If you have any questions or want to discuss competitions/hardware/games/anything with other Kagglers, then join the KaggleNoobs Slack channel here. We also have regular AMAs with top Kagglers there.

And as always, if this helped you, some upvotes would be very much appreciated - that's where I get my motivation! :D

Let's dive right into the data!

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import gc
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

pal = sns.color_palette()

print('# File sizes')
for f in os.listdir('../input'):
    if 'zip' not in f:
        print(f.ljust(30) + str(round(os.path.getsize('../input/' + f) / 1000000, 2)) + 'MB')
```

```
# File sizes
train.csv                      63.4MB
test.csv                       314.02MB
```

Looks like we are simply given two files this time round, one for the training set and one for the test set. They are relatively small compared to other recent competitions, weighing in at less than 400MB total.

It's worth noting that there is a lot more testing data than training data. This could be a sign that some of the test data is dummy data designed to deter hand-labelling, and not included in the calculations, like we recently saw in the DSTL competition.

Let's open up one of the datasets.

Training set

In [2]:
```
df_train = pd.read_csv('../input/train.csv')
df_train.head()
```

Out[2]:

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

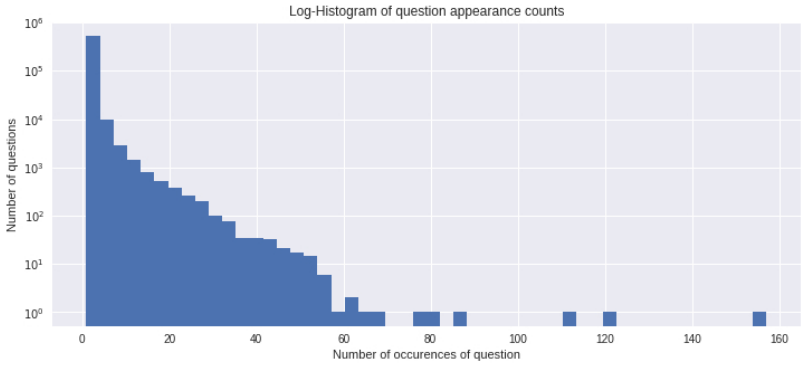
We are given a minimal number of data fields here, consisting of:

`id` : Looks like a simple rowID
`qid{1, 2}` : The unique ID of each question in the pair
`question{1, 2}` : The actual textual contents of the questions.
`is_duplicate` : The **label** that we are trying to predict - whether the two questions are duplicates of each other.

In [3]:

```
print('Total number of question pairs for training: {}'.format(len(df_train)))
print('Duplicate pairs: {}%'.format(round(df_train['is_duplicate'].mean()*100, 2)))
qids = pd.Series(df_train['qid1'].tolist() + df_train['qid2'].tolist())
print('Total number of questions in the training data: {}'.format(len(
    np.unique(qids))))
print('Number of questions that appear multiple times: {}'.format(np.sum(qids.value_counts() >
1)))

plt.figure(figsize=(12, 5))
plt.hist(qids.value_counts(), bins=50)
plt.yscale('log', nonposy='clip')
plt.title('Log-Histogram of question appearance counts')
plt.xlabel('Number of occurences of question')
plt.ylabel('Number of questions')
print()
```

```
Total number of question pairs for training: 404290
Duplicate pairs: 36.92%
Total number of questions in the training data: 537933
Number of questions that appear multiple times: 111780
```



In terms of questions, everything looks as I would expect here. Most questions only appear a few times, with very few questions appearing several times (and a few questions appearing many times). One question appears more than 160 times, but this is an outlier.

We can see that we have a 37% positive class in this dataset. Since we are using the LogLoss metric, and LogLoss looks at the actual predicts as opposed to the order of predictions, we should be able to get a decent score by creating a submission predicting the mean value of the label.

Test Submission

In [4]:

```
from sklearn.metrics import log_loss

p = df_train['is_duplicate'].mean() # Our predicted probability
print('Predicted score:', log_loss(df_train['is_duplicate'], np.zeros_like(df_train['is_duplica
te']) + p))

df_test = pd.read_csv('../input/test.csv')
sub = pd.DataFrame({'test_id': df_test['test_id'], 'is_duplicate': p})
sub.to_csv('naive_submission.csv', index=False)
sub.head()
```

```
Predicted score: 0.658527383984
```

Out[4]:

	is_duplicate	test_id
0	0.369198	0
1	0.369198	1
2	0.369198	2

3	0.369198	3
4	0.369198	4

0.55 on the leaderboard! Score!

However, not all is well. The discrepancy between our local score and the LB one indicates that the distribution of values on the leaderboard is very different to what we have here, which could cause problems with validation later on in the competition.

According to this excellent notebook by David Thaler, using our score and submission we can calculate that we have about 16.5% positives in the test set. This is quite surprising to see, so it'll be something that will need to be taken into account in machine learning models.

Next, I'll take a quick peek at the statistics of the test data before we look at the text itself.

Test Set

In [5]:
```
df_test = pd.read_csv('../input/test.csv')
df_test.head()
```

Out[5]:

	test_id	question1	question2
0	0	How does the Surface Pro himself 4 compare wit...	Why did Microsoft choose core m3 and not core ...
1	1	Should I have a hair transplant at age 24? How...	How much cost does hair transplant require?
2	2	What but is the best way to send money from Ch...	What you send money to China?
3	3	Which food not emulsifiers?	What foods fibre?
4	4	How "aberystwyth" start reading?	How their can I start reading?

In [6]:
```
print('Total number of question pairs for testing: {}'.format(len(df_test)))
```

```
Total number of question pairs for testing: 2345796
```

Nothing out of the ordinary here. We are once again given rowIDs and the textual data of the two questions. It is worth noting that we are not given question IDs here however for the two questions in the pair.

It is also worth pointing out that the actual number of test rows are likely to be much lower than 2.3 million. According to the data page, most of the rows in the test set are using auto-generated questions to pad out the dataset, and deter any hand-labelling. This means that the true number of rows that are scored could be very low.

We can actually see in the head of the test data that some of the questions are obviously auto-generated, as we get delights such as "How their can I start reading?" and "What foods fibre?". Truly insightful questions.

Now onto the good stuff - the text data!

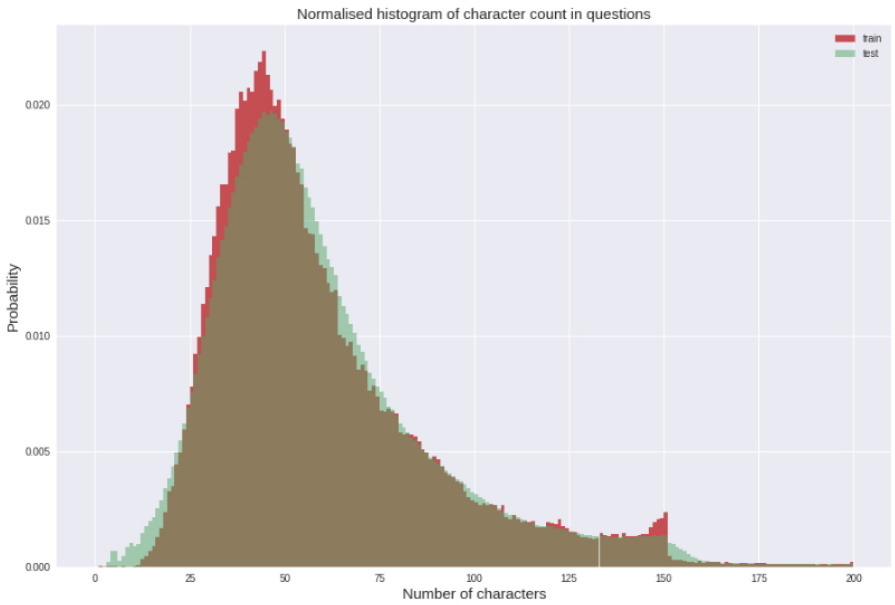
Text analysis

First off, some quick histograms to understand what we're looking at. **Most analysis here will be only on the training set, to avoid the auto-generated questions**

In [7]:
```
train_qs = pd.Series(df_train['question1'].tolist() + df_train['question2'].tolist()).astype(st
r)
test_qs = pd.Series(df_test['question1'].tolist() + df_test['question2'].tolist()).astype(str)

dist_train = train_qs.apply(len)
dist_test = test_qs.apply(len)
plt.figure(figsize=(15, 10))
plt.hist(dist_train, bins=200, range=[0, 200], color=pal[2], normed=True, label='train')
plt.hist(dist_test, bins=200, range=[0, 200], color=pal[1], normed=True, alpha=0.5, label='tes
t')
plt.title('Normalised histogram of character count in questions', fontsize=15)
plt.legend()
plt.xlabel('Number of characters', fontsize=15)
plt.ylabel('Probability', fontsize=15)

print('mean-train {:.2f} std-train {:.2f} mean-test {:.2f} std-test {:.2f} max-train {:.2f} max
-test {:.2f}'.format(dist_train.mean(),
                        dist_train.std(), dist_test.mean(), dist_test.std(), dist_train.max
```

```
(), dist_test.max()))
```

```
mean-train 59.82 std-train 31.96 mean-test 60.07 std-test 31.62 max-train 1169.00 max-test
1176.00
```



We can see that most questions have anywhere from 15 to 150 characters in them. It seems that the test distribution is a little different from the train one, but not too much so (I can't tell if it is just the larger data reducing noise, but it also seems like the distribution is a lot smoother in the test set).

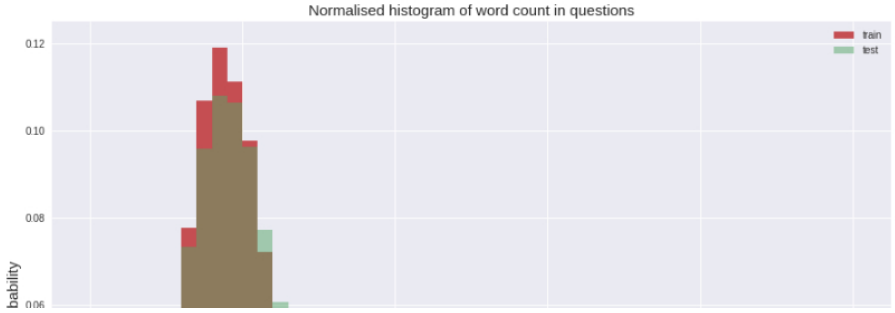
One thing that catches my eye is the steep cut-off at 150 characters for the training set, for most questions, while the test set slowly decreases after 150. Could this be some sort of Quora question size limit?

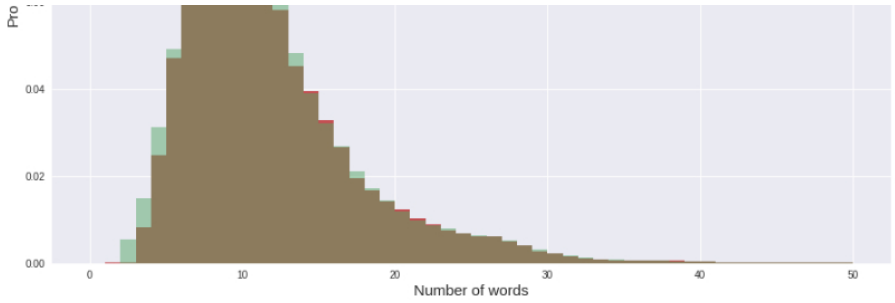
It's also worth noting that I've truncated this histogram at 200 characters, and that the max of the distribution is at just under 1200 characters for both sets - although samples with over 200 characters are very rare.

Let's do the same for word count. I'll be using a naive method for splitting words (splitting on spaces instead of using a serious tokenizer), although this should still give us a good idea of the distribution.

In [8]:
```
dist_train = train_qs.apply(lambda x: len(x.split(' ')))
dist_test = test_qs.apply(lambda x: len(x.split(' ')))

plt.figure(figsize=(15, 10))
plt.hist(dist_train, bins=50, range=[0, 50], color=pal[2], normed=True, label='train')
plt.hist(dist_test, bins=50, range=[0, 50], color=pal[1], normed=True, alpha=0.5, label='test')
plt.title('Normalised histogram of word count in questions', fontsize=15)
plt.legend()
plt.xlabel('Number of words', fontsize=15)
plt.ylabel('Probability', fontsize=15)

print('mean-train {:.2f} std-train {:.2f} mean-test {:.2f} std-test {:.2f} max-train {:.2f} max
-test {:.2f}'.format(dist_train.mean(),
                          dist_train.std(), dist_test.mean(), dist_test.std(), dist_train.max
(), dist_test.max()))
```

```
mean-train 11.06 std-train 5.89 mean-test 11.02 std-test 5.84 max-train 237.00 max-test 23
8.00
```





We see a similar distribution for word count, with most questions being about 10 words long. It looks to me like the distribution of the training set seems more "pointy", while on the test set it is wider. Nevertheless, they are quite similar.

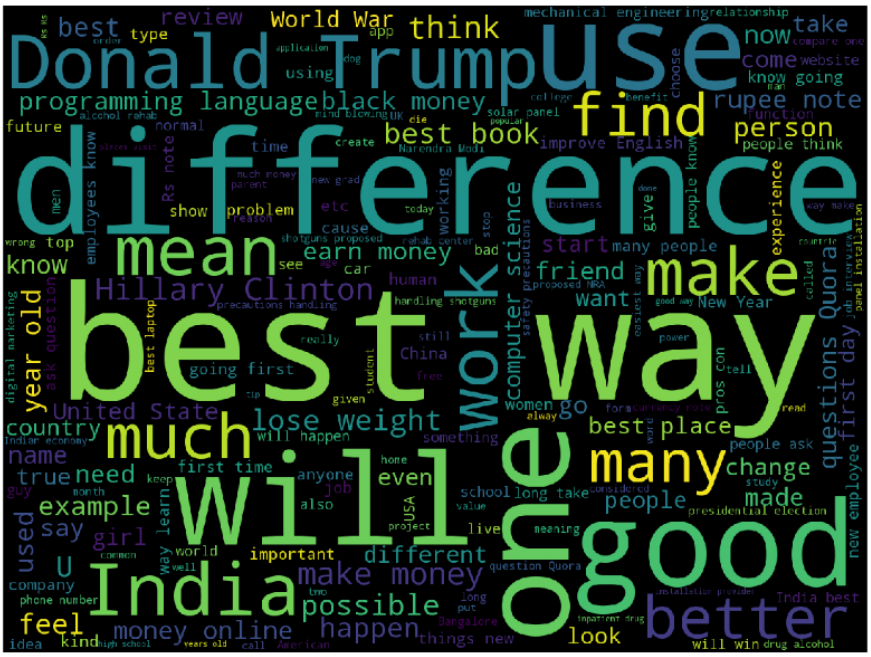
So what are the most common words? Let's take a look at a word cloud.

In [9]:

```
from wordcloud import WordCloud
cloud = WordCloud(width=1440, height=1080).generate(" ".join(train_qs.astype(str)))
plt.figure(figsize=(20, 15))
plt.imshow(cloud)
plt.axis('off')
```

Out[9]:

```
(-0.5, 1439.5, 1079.5, -0.5)
```



Semantic Analysis

Next, I will take a look at usage of different punctuation in questions - this may form a basis for some interesting features later on.

In [10]:

```
qmarks = np.mean(train_qs.apply(lambda x: '?' in x))
math = np.mean(train_qs.apply(lambda x: '[math]' in x))
fullstop = np.mean(train_qs.apply(lambda x: '.' in x))
capital_first = np.mean(train_qs.apply(lambda x: x[0].isupper()))
capitals = np.mean(train_qs.apply(lambda x: max([y.isupper() for y in x])))
numbers = np.mean(train_qs.apply(lambda x: max([y.isdigit() for y in x])))

print('Questions with question marks: {:.2f}%'.format(qmarks * 100))
print('Questions with [math] tags: {:.2f}%'.format(math * 100))
print('Questions with full stops: {:.2f}%'.format(fullstop * 100))
print('Questions with capitalised first letters: {:.2f}%'.format(capital_first * 100))
print('Questions with capital letters: {:.2f}%'.format(capitals * 100))
print('Questions with numbers: {:.2f}%'.format(numbers * 100))
```

```
Questions with question marks: 99.87%
Questions with [math] tags: 0.12%
Questions with full stops: 6.31%
Questions with capitalised first letters: 99.81%
Questions with capital letters: 99.95%
Questions with numbers: 11.83%
```

Initial Feature Analysis

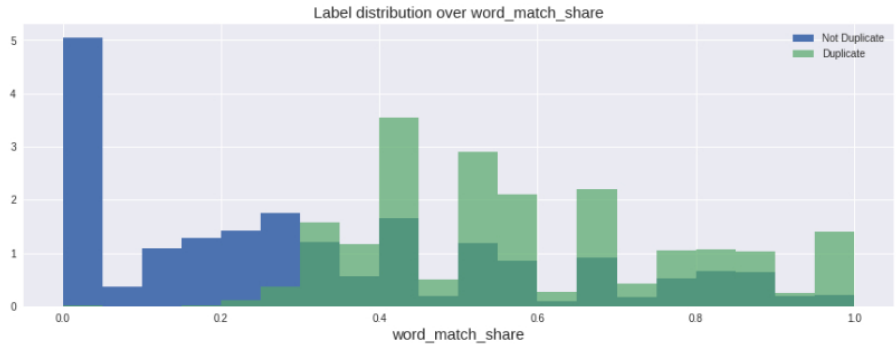
Before we create a model, we should take a look at how powerful some features are. I will start off with the word share feature from the benchmark model.

In [11]:

```
from nltk.corpus import stopwords

stops = set(stopwords.words("english"))

def word_match_share(row):
    q1words = {}
    q2words = {}
    for word in str(row['question1']).lower().split():
        if word not in stops:
            q1words[word] = 1
    for word in str(row['question2']).lower().split():
        if word not in stops:
            q2words[word] = 1
    if len(q1words) == 0 or len(q2words) == 0:
        # The computer-generated chaff includes a few questions that are nothing but stopwords
        return 0
    shared_words_in_q1 = [w for w in q1words.keys() if w in q2words]
    shared_words_in_q2 = [w for w in q2words.keys() if w in q1words]
    R = (len(shared_words_in_q1) + len(shared_words_in_q2))/(len(q1words) + len(q2words))
    return R

plt.figure(figsize=(15, 5))
train_word_match = df_train.apply(word_match_share, axis=1, raw=True)
plt.hist(train_word_match[df_train['is_duplicate'] == 0], bins=20, normed=True, label='Not Dupl
icate')
plt.hist(train_word_match[df_train['is_duplicate'] == 1], bins=20, normed=True, alpha=0.7, labe
l='Duplicate')
plt.legend()
plt.title('Label distribution over word_match_share', fontsize=15)
plt.xlabel('word_match_share', fontsize=15)
```

Out[11]:

```
<matplotlib.text.Text at 0x7f11740696d8>
```



Here we can see that this feature has quite a lot of predictive power, as it is good at separating the duplicate questions from the non-duplicate ones. Interestingly, it seems very good at identifying questions which are definitely different, but is not so great at finding questions which are definitely duplicates.

TF-IDF

I'm now going to try to improve this feature, by using something called TF-IDF (term-frequency-inverse-document-frequency). This means that we weigh the terms by how **uncommon** they are, meaning that we care more about rare words existing in both questions than common one. This makes sense, as for example we care more about whether the word "exercise" appears in both than the word "and" - as uncommon words will be more indicative of the content.

You may want to look into using sklearn's TfidfVectorizer to compute weights if you are implementing this yourself, but as I

You may want to look into using sklearn's TfidfVectorizer to compute weights if you are implementing this yourself, but as I am too lazy to read the documentation I will write a version in pure python with a few changes which I believe should help the score.

In [12]:
```
from collections import Counter

# If a word appears only once, we ignore it completely (likely a typo)
# Epsilon defines a smoothing constant, which makes the effect of extremely rare words smaller
def get_weight(count, eps=10000, min_count=2):
    if count < min_count:
        return 0
    else:
        return 1 / (count + eps)

eps = 5000
words = (" ".join(train_qs)).lower().split()
counts = Counter(words)
weights = {word: get_weight(count) for word, count in counts.items()}
```

In [13]:
```
print('Most common words and weights: \n')
print(sorted(weights.items(), key=lambda x: x[1] if x[1] > 0 else 9999)[:10])
print('\nLeast common words and weights: ')
(sorted(weights.items(), key=lambda x: x[1], reverse=True)[:10])
```

Out[13]:
```
Most common words and weights:

[('the', 2.5891040146646852e-06), ('what', 3.115623919267953e-06), ('is', 3.586170292882527
7e-06), ('how', 4.366449945201053e-06), ('i', 4.480587531263305e-06), ('a', 4.540645588989
843e-06), ('to', 4.671434644293609e-06), ('in', 4.884625153865692e-06), ('of', 5.9202424931
32519e-06), ('do', 6.070908207867897e-06)]

Least common words and weights:

[('シ', 9.998000399920016e-05),
 ('し?', 9.998000399920016e-05),
 ('19-year-old.', 9.998000399920016e-05),
 ('1-855-425-3768', 9.998000399920016e-05),
 ('confederates', 9.998000399920016e-05),
 ('asahi', 9.998000399920016e-05),
 ('fab', 9.998000399920016e-05),
 ('109?', 9.998000399920016e-05),
 ('samrudi', 9.998000399920016e-05),
 ('fulfill?', 9.998000399920016e-05)]
```

In [14]:
```
def tfidf_word_match_share(row):
    q1words = {}
    q2words = {}
    for word in str(row['question1']).lower().split():
        if word not in stops:
            q1words[word] = 1
    for word in str(row['question2']).lower().split():
        if word not in stops:
            q2words[word] = 1
    if len(q1words) == 0 or len(q2words) == 0:
        # The computer-generated chaff includes a few questions that are nothing but stopwords
        return 0

    shared_weights = [weights.get(w, 0) for w in q1words.keys() if w in q2words] + [weights.get
(w, 0) for w in q2words.keys() if w in q1words]
    total_weights = [weights.get(w, 0) for w in q1words] + [weights.get(w, 0) for w in q2words]

    R = np.sum(shared_weights) / np.sum(total_weights)
    return R
```

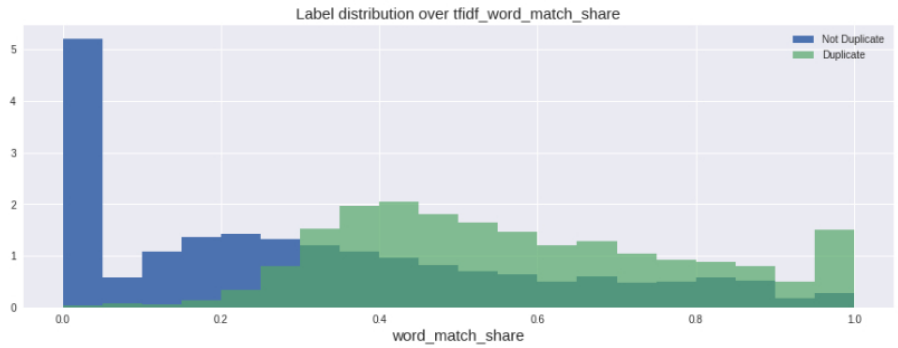
In [15]:
```
plt.figure(figsize=(15, 5))
tfidf_train_word_match = df_train.apply(tfidf_word_match_share, axis=1, raw=True)
plt.hist(tfidf_train_word_match[df_train['is_duplicate'] == 0].fillna(0), bins=20, normed=True,
 label='Not Duplicate')
plt.hist(tfidf_train_word_match[df_train['is_duplicate'] == 1].fillna(0), bins=20, normed=True,
 alpha=0.7, label='Duplicate')
plt.legend()
plt.title('Label distribution over tfidf_word_match_share', fontsize=15)
```

```
plt.xlabel('word_match_share', fontsize=15)
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel/__main__.py:17: RuntimeWarning: invalid va
lue encountered in double_scalars
```

Out[15]:

```
<matplotlib.text.Text at 0x7f114ce4b668>
```



In [16]:

```
from sklearn.metrics import roc_auc_score
print('Original AUC:', roc_auc_score(df_train['is_duplicate'], train_word_match))
print('   TFIDF AUC:', roc_auc_score(df_train['is_duplicate'], tfidf_train_word_match.fillna
(0)))
```

```
Original AUC: 0.780553200628
   TFIDF AUC: 0.77056466105
```

So it looks like our TF-IDF actually got *worse* in terms of overall AUC, which is a bit disappointing. (I am using the AUC metric since it is unaffected by scaling and similar, so it is a good metric for testing the predictive power of individual features.

However, I still think that this feature should provide some extra information which is not provided by the original feature. Our next job is to combine these features and use it to make a prediction. For this, I will use our old friend XGBoost to make a classification model.

Rebalancing the Data

However, before I do this, I would like to rebalance the data that XGBoost receives, since we have 37% positive class in our training data, and only 17% in the test data. By re-balancing the data so our training set has 17% positives, we can ensure that XGBoost outputs probabilities that will better match the data on the leaderboard, and should get a better score (since LogLoss looks at the probabilities themselves and not just the order of the predictions like AUC)

In [17]:

```
# First we create our training and testing data
x_train = pd.DataFrame()
x_test = pd.DataFrame()
x_train['word_match'] = train_word_match
x_train['tfidf_word_match'] = tfidf_train_word_match
x_test['word_match'] = df_test.apply(word_match_share, axis=1, raw=True)
x_test['tfidf_word_match'] = df_test.apply(tfidf_word_match_share, axis=1, raw=True)

y_train = df_train['is_duplicate'].values
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel/__main__.py:17: RuntimeWarning: invalid va
lue encountered in double_scalars
/opt/conda/lib/python3.6/site-packages/ipykernel/__main__.py:17: RuntimeWarning: invalid va
lue encountered in long_scalars
```

In [18]:

```
pos_train = x_train[y_train == 1]
neg_train = x_train[y_train == 0]

# Now we oversample the negative class
# There is likely a much more elegant way to do this...
p = 0.165
scale = ((len(pos_train) / (len(pos_train) + len(neg_train))) / p) - 1
while scale > 1:
    neg_train = pd.concat([neg_train, neg_train])
    scale -=1
```

```
neg_train = pd.concat([neg_train, neg_train[:int(scale * len(neg_train))]])
print(len(pos_train) / (len(pos_train) + len(neg_train)))

x_train = pd.concat([pos_train, neg_train])
y_train = (np.zeros(len(pos_train)) + 1).tolist() + np.zeros(len(neg_train)).tolist()
del pos_train, neg_train
```

```
0.19124366100096607
```

In [19]:

```
# Finally, we split some of the data off for validation
from sklearn.cross_validation import train_test_split

x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train, test_size=0.2, random_s
tate=4242)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/cross_validation.py:43: DeprecationWarning:
This module was deprecated in version 0.18 in favor of the model_selection module into whic
h all the refactored classes and functions are moved. Also note that the interface of the n
ew CV iterators are different from that of this module. This module will be removed in 0.2
0.
  "This module will be removed in 0.20.", DeprecationWarning)
```

XGBoost

Now we can finally run XGBoost on our data, in order to see the score on the leaderboard!

In [20]:

```
import xgboost as xgb

# Set our parameters for xgboost
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 4

d_train = xgb.DMatrix(x_train, label=y_train)
d_valid = xgb.DMatrix(x_valid, label=y_valid)

watchlist = [(d_train, 'train'), (d_valid, 'valid')]

bst = xgb.train(params, d_train, 400, watchlist, early_stopping_rounds=50, verbose_eval=10)
```

```
[0]     train-logloss:0.683189  valid-logloss:0.683238
Multiple eval metrics have been passed: 'valid-logloss' will be used for early stopping.

Will train until valid-logloss hasn't improved in 50 rounds.
[10]    train-logloss:0.602041  valid-logloss:0.602515
[20]    train-logloss:0.544863  valid-logloss:0.545663
[30]    train-logloss:0.503151  valid-logloss:0.504194
[40]    train-logloss:0.471989  valid-logloss:0.473231
[50]    train-logloss:0.448341  valid-logloss:0.449746
[60]    train-logloss:0.430164  valid-logloss:0.431706
[70]    train-logloss:0.416072  valid-logloss:0.417726
[80]    train-logloss:0.405012  valid-logloss:0.406775
[90]    train-logloss:0.396327  valid-logloss:0.398177
[100]   train-logloss:0.389488  valid-logloss:0.391404
[110]   train-logloss:0.384071  valid-logloss:0.386038
[120]   train-logloss:0.379786  valid-logloss:0.381792
[130]   train-logloss:0.376375  valid-logloss:0.378414
[140]   train-logloss:0.373661  valid-logloss:0.375724
[150]   train-logloss:0.371482  valid-logloss:0.373567
[160]   train-logloss:0.369728  valid-logloss:0.371835
[170]   train-logloss:0.368327  valid-logloss:0.370453
[180]   train-logloss:0.367186  valid-logloss:0.369324
[190]   train-logloss:0.366276  valid-logloss:0.368424
[200]   train-logloss:0.365539  valid-logloss:0.367697
[210]   train-logloss:0.364945  valid-logloss:0.367108
[220]   train-logloss:0.364443  valid-logloss:0.366615
[230]   train-logloss:0.364033  valid-logloss:0.366215
[240]   train-logloss:0.363691  valid-logloss:0.365883
[250]   train-logloss:0.363389  valid-logloss:0.3656
[260]   train-logloss:0.363117  valid-logloss:0.365335
```

```
[270]   train-logloss:0.362879  valid-logloss:0.365108
[280]   train-logloss:0.362655  valid-logloss:0.364893
[290]   train-logloss:0.362455  valid-logloss:0.364698
[300]   train-logloss:0.362288  valid-logloss:0.36454
[310]   train-logloss:0.362151  valid-logloss:0.364412
[320]   train-logloss:0.362029  valid-logloss:0.364298
[330]   train-logloss:0.361896  valid-logloss:0.364174
[340]   train-logloss:0.361792  valid-logloss:0.364075
[350]   train-logloss:0.361691  valid-logloss:0.363984
[360]   train-logloss:0.361568  valid-logloss:0.36387
[370]   train-logloss:0.361455  valid-logloss:0.363768
[380]   train-logloss:0.361353  valid-logloss:0.363673
[390]   train-logloss:0.361258  valid-logloss:0.363586
[399]   train-logloss:0.361186  valid-logloss:0.363524
```

In [21]:

```
d_test = xgb.DMatrix(x_test)
p_test = bst.predict(d_test)

sub = pd.DataFrame()
sub['test_id'] = df_test['test_id']
sub['is_duplicate'] = p_test
sub.to_csv('simple_xgb.csv', index=False)
```

0.35460 on the leaderboard - a good first score!

Data

Data Sources

- Quora Question Pairs
  - sample_submission.csv
  - test.csv — 3 columns
  - test.csv
  - train.csv

Quora Question Pairs

Can you identify question pairs that have the same intent?

Last Updated: 3 years ago

About this Competition

The goal of this competition is to predict which of the provided pairs of questions contain two questions with the same meaning. The ground truth is the set of labels that have been supplied by human experts. The ground truth labels are inherently subjective, as the true meaning of sentences can never be known with certainty. Human labeling is also a 'noisy' process, and reasonable people will disagree. As a result, the ground truth labels on this dataset should be taken to be 'informed' but not 100% accurate, and may include incorrect labeling. We believe the labels, on the whole, to represent a reasonable consensus, but this may often not be true on a case by case basis for individual items in the dataset.

Please note: as an anti-cheating measure, Kaggle has supplemented the test set with computer-generated question pairs. Those rows do not come from Quora, and are not counted in the scoring. All of the questions in the training set are genuine examples from Quora.

Data fields

- id - the id of a training set question pair
- qid1, qid2 - unique ids of each question (only available in train.csv)
- question1, question2 - the full text of each question
- is_duplicate - the target variable, set to 1 if question1 and question2 have essentially the same meaning, and 0 otherwise.

Output Files

New Dataset | New Notebook | Download All

Output Files

- naive_submission.csv
- simple_xgb.csv

About this file

This file was created from a Kernel, it does not have a description.

naive_submission.csv

	is_duplicate	test_id
2	0.369197853026293	0
3	0.369197853026293	1
4	0.369197853026293	2
5	0.369197853026293	3
6	0.369197853026293	4
7	0.369197853026293	5
8	0.369197853026293	6
9	0.369197853026293	7
10	0.369197853026293	8
11	0.369197853026293	9
12	0.369197853026293	10
13	0.369197853026293	11
14	0.369197853026293	12
15	0.369197853026293	13
16	0.369197853026293	14
17	0.369197853026293	15
18	0.369197853026293	16
19	0.369197853026293	17
20	0.36919785	18

Run Info

Succeeded	True	Run Time	4.7 seconds
Exit Code	0	Queue Time	0 seconds
Docker Image Name	gcr.io/kaggle-images/python (Dockerfile)	Output Size	0
Timeout Exceeded	False	Used All Space	False
Failure Message			

Log

Download Log

```
Time  Line #  Log Message
           1  [{
           2    "data": "[NbConvertApp] Converting notebook __notebook_source__.ipynb to html\n",
           3    "stream_name": "stderr",
           4    "time": 1.7336498399963602
           5  },{
           6    "data": "[NbConvertApp] Writing 325189 bytes to __results__.html\n",
           7    "stream_name": "stderr",
           8    "time": 1.9221762330271304
           9  }{
          10    "data": "[NbConvertApp] Converting notebook __notebook_source__.ipynb to notebook\n",
```

```
  "stream_name": "stderr",
  "time": 1.7883307879092172
},{
  "data": "[NbConvertApp] Executing notebook with kernel: python3\n",
  "stream_name": "stderr",
  "time": 1.849688916001469
},{
  "data": "[NbConvertApp] Writing 618387 bytes to __notebook__.ipynb\n",
  "stream_name": "stderr",
  "time": 817.026951949927
}{
  "data": "[NbConvertApp] Converting notebook __notebook__.ipynb to html\n",
  "stream_name": "stderr",
  "time": 2.066077855997719
},{
  "data": "[NbConvertApp] Support files will be in __results___files/\n[NbConvertApp] Making directory __results___files\n",
  "stream_name": "stderr",
  "time": 2.310037164017558
},{
  "data": "[NbConvertApp] Making directory __results___files\n[NbConvertApp] Making directory __results___files\n[NbConvertApp] Making directory __results___files\n[NbConvertApp] Making directory __results___files\n[NbConvertApp] Making directory __results___files\n[NbConvertApp] Writing 338580 bytes to __results__.html\n",
  "stream_name": "stderr",
  "time": 2.3149457540130243
}

Complete. Exited with code 0.
```


Comments (158)

Sort by

All Comments | Newest

Click to comment. Be patient, be friendly, and focus on ideas. We are all here to learn and improve!

Ygor G • Posted on Latest Version • 15 days ago • Options • Reply

0

That's a hude kernel! thx

Pratik Pendse • Posted on Latest Version • 16 days ago • Options • Reply

0

Hi All , I am new to NLP.
Can anyone please explain what exactly is happening in word*match*significance method ?
Thanks

daBawse • Posted on Latest Version • 2 months ago • Options • Reply

0

Good stuff

Evan • Posted on Latest Version • 3 months ago • Options • Reply

0

Thanks!

Ramesh Battu • Posted on Latest Version • 3 months ago • Options • Reply

0

Thank you for the notebook, its very useful data analysis intro

Sharp • Posted on Latest Version • 4 months ago • Options • Reply

0

Nicely Done !!

欧阳逸云 • Posted on Latest Version • 8 months ago • Options • Reply

0

Nice job

fiskehandleren • Posted on Latest Version • 10 months ago • Options • Reply

0

Great kernel

DongHuAlonso • Posted on Latest Version • a year ago • Options • Reply 0

Learned a lot, thanks!

Charudatt Chavan • Posted on Latest Version • a year ago • Options • Reply 0

Stuck at XGB iterations...please help.

Gluxable • Posted on Latest Version • a year ago • Options • Reply 0

Hello, I am trying to get this to work on my laptop (just copy pasting) and everything works as expected until word*match*share. I copy paste the In[11] but the output is totally wrong. I have all the modules imported and everything worked up until In[11]. Could I be missing something?

KunLin Lee • Posted on Latest Version • a year ago • Options • Reply 0

This is very helpful kernel for understand the question, thanks for sharing.

ilufei • Posted on Latest Version • a year ago • Options • Reply 0

nice job! thanks for sharing.

Ragav • Posted on Latest Version • a year ago • Options • Reply 0

Hi I am getting this error:

IndexError: only integers, slices (:), ellipsis (...), numpy.newaxis (None) and integer or boolean arrays are valid indices

at this code line: x*test['word*match'] = df*test.apply(word*match_share, axis=1, raw=True)

could you please check

sunpenglv • Posted on Latest Version • a year ago • Options • Reply 0

The type of df*train's rows put into word*matchs*hare and tfidf*word*match*share are different from df*test's, the former is pandas.core.series.Series and the latter is numpy.ndarray, so you need to change row['question1'] into row[1] and row['question2'] into row[2] for df*test.

Ihor Mak • Posted on Latest Version • 2 years ago • Options • Reply 0

Nice work, @anokas!

RenilJoseph • Posted on Latest Version • 2 years ago • Options • Reply 1

Please help , i am getting the following error :
7 q2words = {}
-----> 8 print("", row['question1'])
9 for word in str(row['question1']).lower().split():

IndexError: ('only integers, slices (`:`), ellipsis (`...`), numpy.newaxis (`None`) and integer or boolean arrays are valid indices', 0)

Code which i ran : x*test['word*match'] = df*test.apply(word*match_share, axis=1, raw=True)

Ben Greenwalt • Posted on Latest Version • a year ago • Options • Reply 1

I am having the same problem. Did you find a solution?

Ragav • Posted on Latest Version • a year ago • Options • Reply 2

me too having this issue, can some one please help

Ragav • Posted on Latest Version • a year ago • Options • Reply 0

Hey I am able to run this on Jupyter Notebook.

sunpenglv • Posted on Latest Version • a year ago • Options • Reply

3

The type of df*train's rows put into word*match*share and tfidf*word*match*share are different from df*test's, the former is pandas.core.series.Series and the latter is numpy.ndarray, so you need to change row['question1'] into row[1] and row['question2'] into row[2] for df*test.

Gilbert Yuan • Posted on Latest Version • a year ago • Options • Reply

0

Hi @RenilJoseph
I found the way to proceed successfully, which is simply adding `try catch` in both `word_match_share` and `tfidf_word_match_share` and if catching then `return 0`. Although I can't quite figure out what makes that error which I guess probably due to the `pandas.apply()` version incompatibility, the answer you will get following is exactly the same as anokas's.

ReckonForest • Posted on Latest Version • 2 years ago • Options • Reply

0

Great start!

jaiminpatel • Posted on Latest Version • 2 years ago • Options • Reply

0

Really nice explaination to work with sentences datasets. We can still improve it using NLTK wordnet and gensim models.

Gooner • Posted on Latest Version • 2 years ago • Options • Reply

0

Nice job !!!!

AdityaTodkar • Posted on Latest Version • 2 years ago • Options • Reply

0

@anokas I am new to xgboost I would like to know what is the difference between XGBClassifier() and xgb.DMatrix ? Which will give higher accuracy?

zhangzhm • Posted on Latest Version • 2 years ago • Options • Reply

0

nice notebook :D

guotong1988 • Posted on Latest Version • 2 years ago • Options • Reply

0

If I add some features and then logloss increase to 0.4 , so there must be some problem with the added features?
But AUC and F1 increase..

Nikhil Chaudhari • Posted on Latest Version • 2 years ago • Options • Reply

0

thanks for sharing!

shubzz99 • Posted on Latest Version • 2 years ago • Options • Reply

-1

github

Duo An • Posted on Latest Version • 2 years ago • Options • Reply

0

Nice job,

I add more feature is about `4W-1H` (what why when where how) question word

```
how = np.mean(trainqs.apply(lambda x: 'how' in x.lower()))
why = np.mean(trainqs.apply(lambda x: 'why' in x.lower()))

what = np.mean(trainqs.apply(lambda x: 'what' in x.lower()))
when = np.mean(trainqs.apply(lambda x: 'when' in x.lower()))

where = np.mean(train_qs.apply(lambda x: 'where' in x.lower()))

print('Question contains HOW: {:.2f}%'.format(how * 100))

print('Question contains WHY: {:.2f}%'.format(why * 100))
```

```
print('Question contains WHAT: {:.2f}%'.format(what * 100))

print('Question contains WHEN: {:.2f}%'.format(when * 100))

print('Question contains WHERE: {:.2f}%'.format(where * 100))
```

Result is that:

```
Question contains HOW: 27.26%

Question contains WHY: 10.12%

Question contains WHAT: 39.40%

Question contains WHEN 3.20%

Question contains WHERE: 2.23%
```

About 66.66% question about `how` and `what`.
Question contains 4W-1H words cover 82.1% questions in training dataset, I think it's very very interesting feature.

GSD • Posted on Latest Version • 2 years ago • Options • Reply

0

Thanks for sharing ..Excellent work and the python codes are awesome and reproducible..

Roman Peskov • Posted on Latest Version • 2 years ago • Options • Reply

0

It was very useful for me, thank you!

Angadpreet Nagpal • Posted on Latest Version • 2 years ago • Options • Reply

0

Excellente

nicz • Posted on Latest Version • 3 years ago • Options • Reply

0

Awesome!

Venkata Raj Kiran Kollimarla • Posted on Latest Version • 3 years ago • Options • Reply

0

Great work! Thank you!

wwwhihaho • Posted on Latest Version • 3 years ago • Options • Reply

0

thanks!

Shabie Iqbal • Posted on Latest Version • 3 years ago • Options • Reply

2

Excellent tutorial! thank you very much for it. I have a couple of question however:

```
train_word_match = df_train.apply(word_match_share, axis=1, raw=True)
```

How does this piece of code still pass a Series to the function word*match*share when in fact raw=True is supposed to pass a numpy array to the function.

Secondly, how does calculating both,

```
shared_words_in_q1 = [w for w in q1words.keys() if w in q2words]
shared_words_in_q2 = [w for w in q2words.keys() if w in q1words]
```

help since we know from set theory A intersection B is equal to B intersection A.

Could anyone kindly explain that please? Thanks!

nexemjail • Posted on Latest Version • 3 years ago • Options • Reply

0

Thanks for sharing!

ddlee • Posted on Latest Version • 3 years ago • Options • Reply

0

Thanks a lot for helping me begin.

Kyaw • Posted on Latest Version • 3 years ago • Options • Reply

0

Great job Anokas
But I don't know the final data in here https://www.kaggle.com/c/quora-question-pairs/data and what I got is double size of rows in difference. The actual Train Data Size is 404289 and Test Data Size is 1048576 only and the submitted Data must be 2345796.

Paul Larmuseau • Posted on Latest Version • 3 years ago • Options • Reply

0

@anokas
if you have a trainloss of 0.23 and a valid loss 0.44
does this mean that the result will be 0.44 on the LB ?
or is the logloss not a forecast of the LB logloss ?

[775] train-logloss:0.233366 valid-logloss:0.441991
[800] train-logloss:0.229051 valid-logloss:0.44186
[825] train-logloss:0.225799 valid-logloss:0.442018
[850] train-logloss:0.221929 valid-logloss:0.442329

guotong1988 • Posted on Latest Version • 2 years ago • Options • Reply

0

Same question, could you please answer your own question? Thank you!

SriramanR.Krishnamurthy • Posted on Latest Version • 3 years ago • Options • Reply

0

Thanks for Sharing. This has been a great learning for me !!

Shazan Jabbar • Posted on Latest Version • 3 years ago • Options • Reply

0

Indeed a good starting point with data exploration! Thanks.

prashanth • Posted on Latest Version • 3 years ago • Options • Reply

0

Thanks for sharing. Nice work.

prashanth • Posted on Latest Version • 3 years ago • Options • Reply

0

Thanks for sharing. Nice work.

Head_Cow • Posted on Latest Version • 3 years ago • Options • Reply

0

Good work. But it will be better change int to float in some place.

Lie He • Posted on Latest Version • 3 years ago • Options • Reply

1

remove punctuation is important because for example in row 260393
question1 : Why is Kejriwal against Modi?
question2 : Why is against Kejriwal?
should be obvious the same for this problem but it terms out to be 0 in this feature as "modi" is different from "modi?"

caomaocao • Posted on Latest Version • 3 years ago • Options • Reply

1

Great contribution! Thank you.
This is TfidfVectorizer use in tfidf*word*match_share().

```
tfidf_vectorizer = TfidfVectorizer(stop_words='english', min_df=3)
tfidf_matrix = tfidf_vectorizer.fit_transform(train_qs)
feature_names = tfidf_vectorizer.get_feature_names()
dense = tfidf_matrix.todense()
word_index_dict = dict((j, i) for i,j in enumerate(feature_names))
```

```
def tfidf_word_match_share(row):
    q1words = {}
    q2words = {}
    for word in str(row['question1']).lower().split():
```

```
        if word not in stops:
            q1words[word] = 1
    for word in str(row['question2']).lower().split():
        if word not in stops:
            q2words[word] = 1
    if len(q1words) == 0 or len(q2words) == 0:
        return 0
    q1_tfidf = tfidf_vectorizer.transform([" ".join(q1words.keys())])
    q2_tfidf = tfidf_vectorizer.transform([" ".join(q2words.keys())])
    inter = np.intersect1d(q1_tfidf.indices, q2_tfidf.indices)
    shared_weights = 0
    for word_index in inter:
        shared_weights += (q1_tfidf[0, word_index] + q2_tfidf[0, word_index])
    total_weights = q1_tfidf.sum() + q2_tfidf.sum()
    return np.sum(shared_weights) / np.sum(total_weights)
```

Indra Bhattacharya • Posted on Latest Version • 2 years ago • Options • Reply 0

dense = tfidf_matrix.todense()
This step is returning MemoryError.

[Deleted User] • Posted on Latest Version • 3 years ago • Options • Reply 0

Thanks a lot!

ktrueda • Posted on Latest Version • 3 years ago • Options • Reply 0

thanks! nice work.

YantingCao • Posted on Latest Version • 3 years ago • Options • Reply 0

Does your resampling approach leak information into the cross validation data set? In other words, your cross validation set contains data from training set so it improve logloss. Is this a concern?

anokas Kernel Author • Posted on Latest Version • 3 years ago • Options • Reply 0

You're right, it does! My mistake - this definitely would be a concern and something you would want to fix locally. Interesting that my model didn't appear to overfit anyway.

YantingCao • Posted on Latest Version • 3 years ago • Options • Reply 0

Maybe the part of the test data are generated this way!

Isaac Chávez • Posted on Latest Version • 3 years ago • Options • Reply 0

Nice work! thanks for sharing.

Peter Herr • Posted on Latest Version • 3 years ago • Options • Reply 1

In the stopwords section I had to float the 'R' variable in the word*match*share function because I'm using python 2.7.

```
R = float((len(shared_words_in_q1) + len(shared_words_in_q2)))/float((len(q1words) +
len(q2words)))
return R
```

Nevermind, just found a better way to do it.... thanks

```
from __future__ import print_function, division
```

Anyway, thanks for this @anokas !

Moulta92 • Posted on Latest Version • 3 years ago • Options • Reply 0

Thanks :)

Mario Massimo Mazzarella • Posted on Latest Version • 3 years ago • Options • Reply

0

The notebook is useful and clear, thank you!

Showing 50 of 158 comments. Show More