In [223]:
```
import tweepy
import sys
import jsonpickle
import os, json
import pandas as pd
import numpy as np
import sklearn
import sklearn.feature_extraction
import sklearn.model_selection
import sklearn.metrics
import sklearn.naive_bayes
import sklearn.svm
import sklearn.neighbors
import sklearn.neural_network
from sklearn import decomposition
import nltk
import nltk.corpus
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline


consumer_key = "IQmFVolgzzPWxnJPViusVVrEN"
consumer_secret = "3pCrdesJmIMjJiqywImXcr3xdo6oRVbq9pPyQoixjvwBQ9DAZy"
auth = tweepy.AppAuthHandler(consumer_key,consumer_secret)
api = tweepy.API(auth, wait_on_rate_limit=True,wait_on_rate_limit_notify=True)
stopwds = list(nltk.corpus.stopwords.words('english'))

if (not api):
    print ("Can't Authenticate")
    sys.exit(-1)
```

In [251]:
```
def print_score(Ytrue,Ypred):
    s = (sklearn.metrics.precision_score(Ytrue,Ypred, average = "weighted"),
          sklearn.metrics.recall_score(Ytrue,Ypred,average = "weighted"),
          sklearn.metrics.f1_score(Ytrue,Ypred,average = "weighted"))
    print('Precision: {:0.3}\nRecall: {:0.3}\nF-Score: {:0.3}\n'.format(*s))


def fixdata(a):
    lst = []
    for status in a:
        lst.append(status.text)
    final = pd.DataFrame(lst, columns = ["tweets"])
    return final

def getdata(a):
    searchQuery = a  # this is what we're searching for
    lst = []
    maxTweets = 2000 #
    tweetsPerQry = 100
    twts = []
    sinceId = None
    max_id = -1
    tweetCount = 0
    while tweetCount < maxTweets:
        try:
            if (max_id <= 0):
                if (not sinceId):
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry,lang = "en")
                else:
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry,
                                            since_id=sinceId,lang = "en")
            else:
                if (not sinceId):
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry,
                                            max_id=str(max_id - 1),lang = "en")
                else:
                    new_tweets = api.search(q=searchQuery, count=tweetsPerQry,
                                            max_id=str(max_id - 1),
                                            since_id=sinceId,lang = "en")
            if not new_tweets:
                print("No more tweets found")
                break

            twts.extend(new_tweets)
            tweetCount += len(new_tweets)
            max_id = new_tweets[-1].id
        except tweepy.TweepError as e:
            # Just exit if any error
            print("some error : " + str(e))
            break
    return twts

def tester(tweet):
    if tweet.find("#cats") != -1 and tweet.find("#dogs") != -1:
        return 0
    elif tweet.find("#cats") == -1: #dogs
        return 1
    else: #cats
        return 2
```

In [252]:
```
#Extract tweets using the Twitter API as demonstrated in the tutorial for two specific hashtags:
#Collect equal number of tweets for each and at least 4000 tweets total.

cats_df  = fixdata(getdata("#cats"))
dogs_df  = fixdata(getdata("#dogs"))
main_df = cats_df.append(dogs_df)
```

In [253]:
```
#Separate into three classes:  'dog', 'cat', and 'dog_and_cat',
#meaning #dog and not #cat, #cat' and not #dog, and both, respectively.

main_df['classf'] = main_df['tweets'].apply(tester)
bth_df = main_df[main_df['classf'] == 0]
dog_df = main_df[main_df['classf'] == 1]
cat_df = main_df[main_df['classf'] == 2]
```

In [254]:
```
#Then replace all occurrences of #dog and #cat with a placeholder such as "#blah."

bth_txt = [x.replace('#cats',"blah") for x in bth_df['tweets']]
bth_txt =  [x.replace('#dogs',"blah") for x in bth_txt]
cat_txt = [x.replace('#cats',"blah") for x in cat_df['tweets']]
dog_txt = [x.replace('#dogs',"blah") for x in dog_df['tweets']]
```

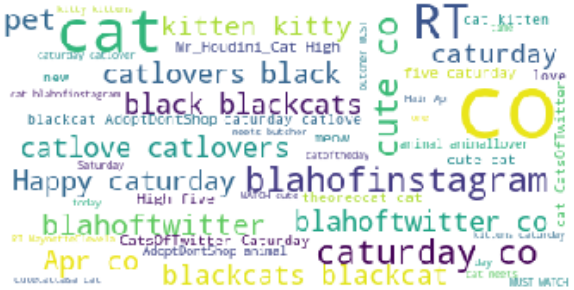
In [267]:
```
vectorizer = sklearn.feature_extraction.text.CountVectorizer(cat_txt+dog_txt+bth_txt, analyzer =
vectorizer.fit(cat_txt+dog_txt+bth_txt)
cat_tdm = vectorizer.transform(cat_txt).toarray()
dog_tdm = vectorizer.transform(dog_txt).toarray()
bth_tdm = vectorizer.transform(bth_txt).toarray()
#vectorizer.get_feature_names()[0]
```

In [264]:
```
from wordcloud import WordCloud, STOPWORDS
stopwords = set(STOPWORDS)

wordcloud = WordCloud(
                          background_color='white',
                          stopwords= stopwords.union({"https","blah"}),
                          max_words=50,
                          max_font_size=60,
                          random_state=10000
                         ).generate(''.join(cat_txt))

print(wordcloud)
fig = plt.figure(1)
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
fig.savefig("cats.png", dpi=900)
```

```
<wordcloud.wordcloud.WordCloud object at 0x0000017839000BE0>
```

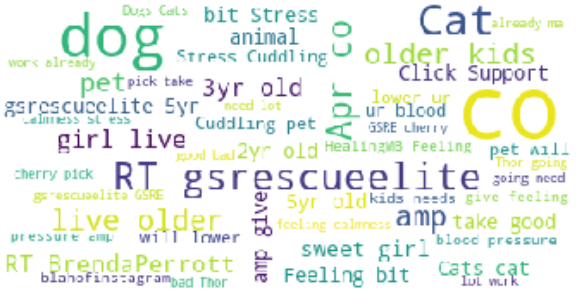


In [265]:
```
from wordcloud import WordCloud, STOPWORDS
stopwords = set(STOPWORDS)

wordcloud = WordCloud(
                            background_color='white',
                            stopwords= stopwords.union({"https","blah"}),
                            max_words=50,
                            max_font_size=60,
                            random_state=10000
                           ).generate(''.join(dog_txt))

print(wordcloud)
fig = plt.figure(1)
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
fig.savefig("dog.png", dpi=900)
```

```
<wordcloud.wordcloud.WordCloud object at 0x0000017839028F98>
```

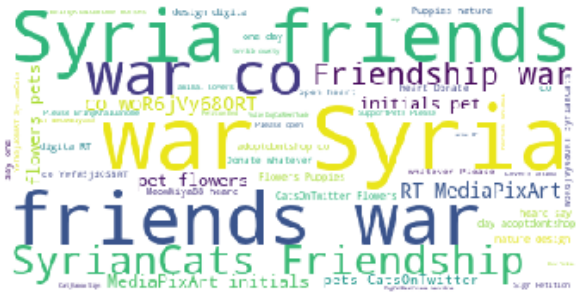


In [266]:
```
from wordcloud import WordCloud, STOPWORDS
stopwords = set(STOPWORDS)

wordcloud = WordCloud(
                            background_color='white',
                            stopwords= stopwords.union({"https","blah"}),
                            max_words=50,
                            max_font_size=60,
                            random_state=10000
                           ).generate(''.join(bth_txt))

print(wordcloud)
fig = plt.figure(1)
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
fig.savefig("both.png", dpi=900)
```

```
<wordcloud.wordcloud.WordCloud object at 0x00000178391F8E10>
```



In [268]:
```
zeros = np.zeros((len(bth_txt),1))
ones = np.ones((len(dog_txt),1))
two  = np.ones((len(cat_txt),1)) + 1

all_tdm = np.concatenate((bth_tdm,dog_tdm,cat_tdm),axis=0)  #there are 1323 features!!!
Y = np.ravel(np.concatenate((zeros,ones,two),axis=0))


#To develop classification models, you will need to extract and select appropriate features.
#These 300 PCA explain 92% of the variance
#This is a reduction of almost 1023 features!
pca = decomposition.PCA(n_components=100)
pca.fit(all_tdm)
print("These PCA explain",sum(pca.explained_variance_ratio_),"of the variance! This is a reductio
all_tdm = pca.transform(all_tdm)
```

```
These PCA explain 0.6556681606025629 of the variance! This is a reduction of 1223 features
```

In [269]:
```
#Split the data for training and testing, approximately 80-20%.
Xtrain,Xtest,Ytrain,Ytest = sklearn.model_selection.train_test_split(all_tdm, Y, test_size=.20)
"""### Naive Bayes"""

nb = sklearn.naive_bayes.GaussianNB()
nb.fit(Xtrain,Ytrain)
Ypred = nb.predict(Xtest)

print("\nNaive Bayes Performance")
print_score(Ytest,Ypred)

"""### Neural Network"""

nn = sklearn.neural_network.MLPClassifier()
nn.fit(Xtrain,Ytrain)
Ypred = nn.predict(Xtest)

print("\nNeural Network Performance")
print_score(Ytest,Ypred)


"""### SVM"""

svm = sklearn.svm.SVC()
svm.fit(Xtrain,Ytrain)
Ypred = svm.predict(Xtest)

print("\nSVM performance")
print_score(Ytest,Ypred)
```

```
Naive Bayes Performance
Precision: 0.722
Recall: 0.603
F-Score: 0.626


Neural Network Performance
Precision: 0.954
Recall: 0.952
F-Score: 0.953


SVM performance
Precision: 0.899
Recall: 0.894
F-Score: 0.895
```

In [ ]:
```

```