In [30]:
```
#Utils
import re
import pprint
import itertools
import numpy as np
import pandas as pd
from time import time
from datetime import datetime
from collections import Counter
import matplotlib.pyplot as plt


#NLP
import nltk
from nltk.corpus import stopwords
import language_check
from spellchecker import SpellChecker
from nltk.tag.perceptron import PerceptronTagger
from gensim.models import word2vec

#ML
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import cohen_kappa_score as kappa
from sklearn.metrics.pairwise import cosine_similarity

import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')

%matplotlib inline
```

Loading and Processing essay data into sets

In [31]:
```
def split_in_sets(data):
    essay_sets = []
    min_scores = []
    max_scores = []
    for s in range(1,9):
        essay_set = data[data["essay_set"] == s]
        essay_set.dropna(axis=1, inplace=True)
        n, d = essay_set.shape
        set_scores = essay_set["domain1_score"]
        print ("Set", s, ": Essays = ", n , "\t Attributes = ", d)
        min_scores.append(set_scores.min())
        max_scores.append(set_scores.max())
        essay_sets.append(essay_set)
    return (essay_sets, min_scores, max_scores)
```

In [32]:
```
dataset_path = "./asap-aes/training_set_rel3.tsv"
data = pd.read_csv(dataset_path, sep="\t", encoding="ISO-8859-1", i

min_scores = [2, 1, 0, 0, 0, 0, 0, 0]
max_scores = [12, 6, 3, 3, 4, 4, 30, 60]

essay_sets, data_min_scores, data_max_scores = split_in_sets(data)
set1, set2, set3, set4, set5, set6, set7, set8 = tuple(essay_sets)
data.dropna(axis=1, inplace=True)

data.drop(columns=["rater1_domain1", "rater2_domain1"], inplace=Tru
print("All Data:", len(data))
data.head()
```

```
Set 1 : Essays =  1783   Attributes =  5
Set 2 : Essays =  1800   Attributes =  8
Set 3 : Essays =  1726   Attributes =  5
Set 4 : Essays =  1770   Attributes =  5
Set 5 : Essays =  1805   Attributes =  5
Set 6 : Essays =  1800   Attributes =  5
Set 7 : Essays =  1569   Attributes =  13
Set 8 : Essays =  723    Attributes =  17
All Data: 12976
```

Out[32]:

essay_id	essay_set	essay	domain1_score
1	1	Dear local newspaper, I think effects computer...	8
2	1	Dear @CAPS1 @CAPS2, I believe that using compu...	9
3	1	Dear, @CAPS1 @CAPS2 @CAPS3 More and more peopl...	7
4	1	Dear Local Newspaper, @CAPS1 I have found that...	10
5	1	Dear @LOCATION1, I know having computers has a...	8

In [33]:
```
# Read prompt text
set_3_prompt = open("./prompts/set3.txt").read()
set_3_prompt = set_3_prompt.splitlines()

set_4_prompt = open("./prompts/set4.txt").read()
set_4_prompt = set_4_prompt.splitlines()

set_5_prompt = open("./prompts/set5.txt").read()
set_5_prompt = set_5_prompt.splitlines()

set_6_prompt = open("./prompts/set6.txt").read()
set_6_prompt = set_6_prompt.splitlines()

set_prompts = [set_3_prompt, set_4_prompt, set_5_prompt, set_6_prom
```

In [34]:
```
print("Minimum Scores: ", min_scores)
print("Maximum Scores: ", max_scores)
```

```
Minimum Scores:  [2, 1, 0, 0, 0, 0, 0, 0]
Maximum Scores:  [12, 6, 3, 3, 4, 4, 30, 60]
```

In [35]:
```
#Dataset keys
essay_id_key = "essay_id"
essay_set_key = "essay_set"
essay_key = "essay"
domain1_score_key = "domain1_score"
```

In [36]:
```
#Feature keys
feature_keys = {
    "char_count_key": "char_count",
    "word_count_key": "word_count",
    "diff_words_count_key": "diff_words_count",
    "word_count_root_key": "word_count_root",
    "sen_count_key": "sen_count",
    "avg_word_len_key": "avg_word_len",
    "avg_sen_len_key": "avg_sen_len",
    "l5_word_count_key": "l5_word_count",
    "l6_word_count_key": "l6_word_count",
    "l7_word_count_key": "l7_word_count",
    "l8_word_count_key": "l8_word_count",
}
```

In [37]:
```
#Extra features
extra_feature_keys = {
    # "grammer_error_count_key": "grammer_error_count",
    "spelling_error_count_key": "spelling_error_count",
    "stopwords_count_key": "stopwords_count",
    "small_sentences_count_key": "small_sentence_count", #sentences
    # "beautiful_words_count_key": "beautiful_words_count",
    "punctuations_count_key": "punctuations_count",
    "verbs_count_key": "verbs_count",
    "adverbs_count_key": "adverbs_count",
    "nouns_count_key": "nouns_count",
    "adjectives_count_key": "adjective_count",
}
```

In [38]:
```
sentences_key = "sentences"
```

In [39]:
```
feature_keys_list = list(feature_keys.values())
extra_feature_keys_list = list(extra_feature_keys.values())
all_feature_keys_list = feature_keys_list + extra_feature_keys_list

print("Basic 11 features: ", feature_keys_list, "\n")
print("Extra features: ", extra_feature_keys_list, "\n")
print("All features: ", all_feature_keys_list, "\n")
```

```
Basic 11 features:  ['char_count', 'word_count', 'diff_words_count
', 'word_count_root', 'sen_count', 'avg_word_len', 'avg_sen_len',
'l5_word_count', 'l6_word_count', 'l7_word_count', 'l8_word_count'
]

Extra features:  ['spelling_error_count', 'stopwords_count', 'smal
l_sentence_count', 'punctuations_count', 'verbs_count', 'adverbs_c
ount', 'nouns_count', 'adjective_count']

All features:  ['char_count', 'word_count', 'diff_words_count', 'w
ord_count_root', 'sen_count', 'avg_word_len', 'avg_sen_len', 'l5_w
ord_count', 'l6_word_count', 'l7_word_count', 'l8_word_count', 'sp
elling_error_count', 'stopwords_count', 'small_sentence_count', 'p
unctuations_count', 'verbs_count', 'adverbs_count', 'nouns_count',
'adjective_count']
```

Feature generation from essaay

In [61]:
```
tagger=PerceptronTagger()
tool = language_check.LanguageTool('en-US')
spell = SpellChecker()
spell.word_frequency.load_words(["PERSON", "ORGANIZATION", "LOCATIO
max_idx = 0
word_to_idx = {}
idx_to_word = {}
```

```
def sentence_to_word_list(sentence, remove_stopwords):
    # Remove non letter from sentenece and stop words
    global max_idx
    global word_to_idx
    global idx_to_word

    sen_char_count = 0
    sen_word_count = 0
    l5_sen_word_count = 0
    l6_sen_word_count = 0
    l7_sen_word_count = 0
    l8_sen_word_count = 0
    sen_diff_words = set()
    ### Extra Features ###
    sen_verbs_count = 0
    sen_adverbs_count = 0
    sen_nouns_count = 0
    sen_adjectives_count = 0
    sen_spelling_error_count = 0
    sen_stopwords_count = 0
    is_small_sentence = 0

    stops = set(stopwords.words("english"))
    all_words = sentence.lower().split()


    kept_words = []

    if len(all_words) <= 4: is_small_sentence = 1

    misspelled = spell.unknown(all_words)
    sen_spelling_error_count = len(misspelled)

    for word in all_words:

        if not word in word_to_idx:
            word_to_idx[word] = max_idx
            idx_to_word[max_idx] = word
            max_idx += 1

        sen_char_count += len(word)
        sen_word_count += 1
        word_len = len(word)
        if word_len > 5:
            l5_sen_word_count += 1
        if word_len > 6:
            l6_sen_word_count += 1
        if word_len > 7:
            l7_sen_word_count += 1
        if word_len > 8:
            l8_sen_word_count += 1

        sen_diff_words.add(word)
```

```
            sen_diff_words.add(word)

            isStopword = word in stops
            if isStopword: sen_stopwords_count += 1

            if remove_stopwords and not isStopword:
                kept_words.append(word)
            else:
                kept_words.append(word)

        features = {
             feature_keys["char_count_key"]: sen_char_count,
             feature_keys["word_count_key"]: sen_word_count,
             feature_keys["l5_word_count_key"]: l5_sen_word_count,
             feature_keys["l6_word_count_key"]: l6_sen_word_count,
             feature_keys["l7_word_count_key"]: l7_sen_word_count,
             feature_keys["l8_word_count_key"]: l8_sen_word_count,
             feature_keys["diff_words_count_key"]: sen_diff_words
        }

        extra_features = {
            extra_feature_keys["small_sentences_count_key"]: is_small_s
            extra_feature_keys["spelling_error_count_key"]: sen_spellin
            extra_feature_keys["stopwords_count_key"]: sen_stopwords_co
            extra_feature_keys["verbs_count_key"]: sen_verbs_count,
            extra_feature_keys["adverbs_count_key"]: sen_adverbs_count,
            extra_feature_keys["nouns_count_key"]: sen_nouns_count,
            extra_feature_keys["adjectives_count_key"]: sen_adjectives_
        }

        return (kept_words, features, extra_features)

    def essay_to_sentences(essay, remove_stopwords = False):
        # Convert essay into sentence


        tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
        sentences = tokenizer.tokenize(essay.strip())
        split_sentences = []
        split_words = []

        char_count = 0
        word_count = 0
        diff_words = set()
        word_count_root = 0
        sen_count = 0
        avg_word_len = 0
        avg_sen_len = 0
        l5_word_count = 0
        l6_word_count = 0
        l7_word_count = 0
        l8_word_count = 0


        ### Extra Features ###
```

```
    ### Extra Features ###
    spelling_error_count = 0
    stopwords_count = 0
    small_sentences_count = 0
    punctuation_count = 0
    grammer_error_count = 0
    small_sentences_count = 0
    verbs_count = 0
    adverbs_count = 0
    nouns_count = 0
    adjectives_count = 0

    all_words = nltk.word_tokenize(essay)
    count= Counter([j for i,j in tagger.tag(all_words)])
    verbs_count = count['VB'] + count['VBG'] + count['VBP'] + count
    adverbs_count = count['RB'] + count['RBR'] + count['RBS']
    nouns_count = count['NN'] + count['NNS'] + count['NNPS'] + coun
    adjectives_count = count['JJ'] + count['JJR']

    punctuation = ['.','?', '!', ':', ';']
    for punct in punctuation:
        punctuation_count += essay.count(punct)

    for sentence in sentences:
        if len(sentence) > 0:
            sentence = re.sub("[^a-zA-Z]", " ", sentence)
            # grammer_error_count += len(tool.check(sentence))


            kept_words, features, extra_features = sentence_to_word
            split_sentences.append(kept_words)
            split_words.extend(kept_words)

            sen_count +=1
            char_count += features[feature_keys["char_count_key"]]
            word_count += features[feature_keys["word_count_key"]]
            l5_word_count += features[feature_keys["l5_word_count_k
            l6_word_count += features[feature_keys["l6_word_count_k
            l7_word_count += features[feature_keys["l7_word_count_k
            l8_word_count += features[feature_keys["l8_word_count_k
            diff_words = diff_words|features[feature_keys["diff_wor
            ### Extra Features ###
            spelling_error_count += extra_features[extra_feature_ke
            stopwords_count += extra_features[extra_feature_keys["s
            small_sentences_count += extra_features[extra_feature_k

    word_count_root = word_count ** (1/4)
    avg_word_len = char_count / word_count
    avg_sen_len = word_count / sen_count

    features = {
        feature_keys["char_count_key"]: char_count,
        feature_keys["word_count_key"]: word_count,
        feature_keys["diff_words_count_key"]: len(diff_words),
        feature_keys["word_count_root_key"]: word_count_root
```

```
        feature_keys["word_count_root_key"]: word_count_root,
        feature_keys["sen_count_key"]: sen_count,
        feature_keys["avg_word_len_key"]: avg_word_len,
        feature_keys["avg_sen_len_key"]: avg_sen_len,
        feature_keys["l5_word_count_key"]: l5_word_count,
        feature_keys["l6_word_count_key"]: l6_word_count,
        feature_keys["l7_word_count_key"]: l7_word_count,
        feature_keys["l8_word_count_key"]: l8_word_count
    }

    extra_features = {
        # extra_feature_keys["grammer_error_count_key"]: grammer_er
        extra_feature_keys["spelling_error_count_key"]: spelling_er
        extra_feature_keys["stopwords_count_key"]: stopwords_count,
        extra_feature_keys["small_sentences_count_key"]: small_sent
        extra_feature_keys["punctuations_count_key"]: punctuation_c
        extra_feature_keys["verbs_count_key"]: verbs_count,
        extra_feature_keys["adverbs_count_key"]: adverbs_count,
        extra_feature_keys["nouns_count_key"]: nouns_count,
        extra_feature_keys["adjectives_count_key"]: adjectives_coun
    }

    return (split_words, split_sentences, features, extra_features)


def get_feature_vec(words, model, num_features):
    featureVec = np.zeros((num_features,),dtype="float32")
    nwords = 0.
    index2word_set = set(model.wv.index2word)
    for word in words:
        if word in index2word_set:
            nwords = nwords + 1.
            featureVec = np.add(featureVec, model[word])
    featureVec = np.divide(featureVec, nwords)
    return featureVec


def get_word_count_vector(words):
    word_count_vector = np.zeros((max_idx+1,))
    for word in words:
        if word in word_to_idx:
            word_count_vector[word_to_idx[word]] += 1
    return word_count_vector

def get_prompt_vectors(prompt):
    whole_prompt_words = []
    vectors = []
    for para in prompt:
        tokenizer = nltk.data.load('tokenizers/punkt/english.pickle
        sentences = tokenizer.tokenize(para.strip())
        para_words = []
        for sentence in sentences:
            if len(sentence) > 0:
                sentence = re.sub("[^a-zA-Z]", " ", sentence)
                words = sentence.lower().split()
```

```
                words = sentence.lower().split()
                para_words.extend(words)
                whole_prompt_words.extend(words)
        vectors.append(get_word_count_vector(para_words))
    vectors.append(get_word_count_vector(para_words))
    return vectors
```

Example of style based features generated

In [62]:
```
pp = pprint.PrettyPrinter(indent=4)

#Featrues
first_essay = data.iloc[0][essay_key]
print(first_essay)
start = time()
_, _, features, extra_features = essay_to_sentences(first_essay)
end = time()

print("\nExecution time:", end-start)

print("\n\nFeatures: ")
pp.pprint(features)

print("\n\n Extra Features: ")
pp.pprint(extra_features)
```

```
Dear local newspaper, I think effects computers have on people are
great learning skills/affects because they give us time to chat wi
th friends/new people, helps us learn about the globe(astronomy) a
nd keeps us out of troble! Thing about! Dont you think so? How wou
ld you feel if your teenager is always on the phone with friends!
Do you ever time to chat with your friends or buisness partner abo
ut things. Well now - there's a new way to chat the computer, thei
rs plenty of sites on the internet to do so: @ORGANIZATION1, @ORGA
NIZATION2, @CAPS1, facebook, myspace ect. Just think now while you
r setting up meeting with your boss on the computer, your teenager
is having fun on the phone not rushing to get off cause you want t
o use it. How did you learn about other countrys/states outside of
yours? Well I have by computer/internet, it's a new way to learn a
bout what going on in our time! You might think your child spends
a lot of time on the computer, but ask them so question about the
economy, sea floor spreading or even about the @DATE1's you'll be
surprise at how much he/she knows. Believe it or not the computer
is much interesting then in class all day reading out of books. If
your child is home on your computer or at a local library, it's be
tter than being out with friends being fresh, or being perpressure
d to doing something they know isnt right. You might not know wher
e your child is, @CAPS2 forbidde in a hospital bed because of a dr
ive-by. Rather than your child on the computer learning, chatting
or just playing games, safe and sound in your home or community pl
ace. Now I hope you have reached a point to understand and agree w
ith me, because computers can have great effects on you or child b
```

```
ecause it gives us time to chat with friends/new people, helps us
learn about the globe and believe or not keeps us out of troble. T
hank you for listening.

Execution time: 0.03949308395385742


Features:
{   'avg_sen_len': 21.875,
    'avg_word_len': 4.222857142857142,
    'char_count': 1478,
    'diff_words_count': 164,
    'l5_word_count': 74,
    'l6_word_count': 59,
    'l7_word_count': 34,
    'l8_word_count': 13,
    'sen_count': 16,
    'word_count': 350,
    'word_count_root': 4.3253077270721105}


 Extra Features:
{   'adjective_count': 22,
    'adverbs_count': 21,
    'nouns_count': 84,
    'punctuations_count': 17,
    'small_sentence_count': 3,
    'spelling_error_count': 9,
    'stopwords_count': 184,
    'verbs_count': 67}
```

In [63]:
```
def makeDataFrame(data):
    all_features = {}
    all_scores = {}

    for row in range(len(data)):
        if row % 1500 == 0 and row != 0: print("Processed ", row, "
        essay_data = data.iloc[row]
        essay = essay_data[essay_key]
        # essay_id = essay_data[essay_id_key]
        essay_id = essay_data.name
        essay_score = essay_data[domain1_score_key]
        words, _, features, extra_features = essay_to_sentences(ess

        combined_features = {}
        combined_features.update(features)
        combined_features.update(extra_features)
        combined_features[sentences_key] = words
        all_features[essay_id] = combined_features
        all_scores[essay_id] = essay_score

    X = pd.DataFrame.from_dict(all_features, orient="index")
    y = pd.DataFrame.from_dict(all_scores, orient="index")

    return(X, y)
```

In [64]:
```
start = time()
X, y = makeDataFrame(data)
end = time()
X_basic = X[feature_keys_list + [sentences_key]]

print("\nExecution time to make dataframe ", (end-start)/60, " mins

print("All Features:\n")
display(X.head(2))

print("Basic 11 Features:\n")
display(X_basic.head(2))
```

```
Processed  1500  essays of 12976  rows.
Processed  3000  essays of 12976  rows.
Processed  4500  essays of 12976  rows.
Processed  6000  essays of 12976  rows.
Processed  7500  essays of 12976  rows.
Processed  9000  essays of 12976  rows.
Processed  10500  essays of 12976  rows.
Processed  12000  essays of 12976  rows.

Execution time to make dataframe  3.0431220014890035  mins

All Features:
```

	char_count	word_count	diff_words_count	word_count_root	sen_count	avg_word_len	a
1	1478	350	164	4.325308	16	4.222857	
2	1814	423	192	4.535081	20	4.288416	

Basic 11 Features:

	char_count	word_count	diff_words_count	word_count_root	sen_count	avg_word_len	a
1	1478	350	164	4.325308	16	4.222857	
2	1814	423	192	4.535081	20	4.288416	

In [83]:
```
X_sets = []
y_sets = []
essay_sets = [set1, set2, set3, set4, set5, set6, set7, set8]
for set_no in range(8):
    # X_set, y_set = makeDataFrame(essay_sets[set_no])
    X_set, y_set = X.loc[X.index & essay_sets[set_no].index], y.loc
    X_sets.append(X_set)
    y_sets.append(y_set)

    print("Set", (set_no+1), ":", len(X_set), " essays")
    display(X_set.head(1))
```

Set 1 : 1783 essays

	char_count	word_count	diff_words_count	word_count_root	sen_count	avg_word_len	a
1	1478	350	164	4.325308	16	4.222857	

Set 2 : 1800 essays

	char_count	word_count	diff_words_count	word_count_root	sen_count	avg_word_ler
	2978	2064	486	194	4.695254	19	4.24691

Set 3 : 1726 essays

	char_count	word_count	diff_words_count	word_count_root	sen_count	avg_word_ler
	5978	209	51	31	2.672345	3	4.09803

Set 4 : 1770 essays

	char_count	word_count	diff_words_count	word_count_root	sen_count	avg_word_ler
	8863	243	58	44	2.759669	3	4.18965

Set 5 : 1805 essays

	char_count	word_count	diff_words_count	word_count_root	sen_count	avg_word_le
	11827	570	131	80	3.383123	8	4.35114

Set 6 : 1800 essays

	char_count	word_count	diff_words_count	word_count_root	sen_count	avg_word_le
	14834	560	123	88	3.330246	6	4.55284

Set 7 : 1569 essays

	char_count	word_count	diff_words_count	word_count_root	sen_count	avg_word_le

17834	382	94	57	3.113737	3	4.0638

Set 8 : 723 essays

	char_count	word_count	diff_words_count	word_count_root	sen_count	avg_word_le
20716	2562	708	239	5.15832	39	3.61864

In [65]:
```
prompt_vectors = []
for set_prompt in set_prompts:
    prompt_vectors.append(get_prompt_vectors(set_prompt))
```

In [87]:
```
def evaluate(X, y, model = LinearRegression(), plot=False, wordvec=

    X, X_unseen, y, y_unseen = train_test_split(X, y, test_size=0.1
    kf = KFold(n_splits=5, shuffle=True)
    cv = kf.split(X)
    results = []

    start = time()
    for traincv, testcv in cv:
            X_test, X_train, y_test, y_train = X.iloc[testcv], X.il

            if wordvec:

                min_word_count = 40
                num_workers = 4
                downsampling = 1e-3
                sentences = X_train[sentences_key]
                wv_model = word2vec.Word2Vec(sentences, workers=num
                wv_model.init_sims(replace=True)

                vec_key = "word vec"
                train_essay_vectors = {}
                test_essay_vectors = {}

                for idx in X_train.index.values:
                    train_essay_vectors[idx] = {"wv_"+str(ind): vec
                X_train_vec = pd.DataFrame.from_dict(train_essay_ve

                for idx in X_test.index.values:
                    test_essay_vectors[idx] = {"wv_"+str(ind): vec
                X_test_vec = pd.DataFrame.from_dict(test_essay_vect

                X_train = X_train.join(X_train_vec)
                X_test = X_test.join(X_test_vec)
```

```
            if not prompt_vecs is None:
                train_essay_similarity = {}
                test_essay_similarity = {}
                labels = ["para 1", "para 2", "para 3", "para 4", "

                for idx in X_train.index.values:
                    similarity = {}
                    essay_vector = get_word_count_vector(X_train.lo
                    for i in range(len(prompt_vecs)):
                        sim = cosine_similarity(prompt_vecs[i].resh
                        similarity[labels[i]] = sim
                    train_essay_similarity[idx] = similarity
                X_train_vec = pd.DataFrame.from_dict(train_essay_ve


                for idx in X_test.index.values:
                    similarity = {}
                    essay_vector = get_word_count_vector(X_test.loc
                    for i in range(len(prompt_vecs)):
                        sim = cosine_similarity(prompt_vecs[i].resh
                        similarity[labels[i]] = sim
                    test_essay_similarity[idx] = similarity
                X_test_vec = pd.DataFrame.from_dict(test_essay_simi

                X_train = X_train.join(X_train_vec)
                X_test = X_test.join(X_test_vec)


            X_train.drop([sentences_key], axis=1, inplace=True)
            X_test.drop([sentences_key], axis=1, inplace=True)


            model.fit(X_train,y_train.values.ravel())


            y_pred = model.predict(X_test)
            y_pred = y_pred.reshape(-1)
            y_pred = np.around(y_pred, decimals=0).astype(int)
            y_test = [item for sublist in y_test.values for item in

            result = kappa(y_test,y_pred,labels=None, weights='quad
            results.append(result)
    end = time()

    print("[", round((end-start)/60, 3), " mins", end=" ] ")

    if plot:
        y_unseen_pred = model.predict(X_unseen)
        y_unseen_pred = y_unseen_pred.reshape(-1)
        y_unseen_pred = np.around(y_unseen_pred, decimals=0).astype

        labels = set(y_unseen_pred)
        ncol = int(len(labels)/2.5)
        plt.figure(figsize=(8, 8))
```

```
        scatter = plt.scatter(X_unseen.index,y_unseen, c=y_unseen_p
        handles = scatter.legend_elements()[0]
        plt.legend(handles=handles, labels=labels, ncol=ncol, loc="
        plt.ylabel("Score")
        plt.xlabel("Essay ID")
        model_name = type(model).__name__
        plt.title("Model: " + model_name)

        dateTimeObj = datetime.now()
        timestampStr = dateTimeObj.strftime("%d-%b-%Y (%H:%M:%S.%f)
        plt.savefig("./figs/" + model_name + "-" +timestampStr + ".

    return np.array(results).mean(), model
```

In [78]:
```
def get_classifiers():
    linr = LinearRegression()
    knn = KNeighborsClassifier(10)
    return [linr, knn]

def get_all_classifiers():
    linr = LinearRegression()
    svm = SVC(kernel="linear", C=0.025)
    knn = KNeighborsClassifier(10)
    return [linr, svm, knn]
```

In [20]:
```
# Training on individual dataset
for classifier in get_all_classifiers():
    print("Basic 11 features ( Model: ", type(classifier).__name__,
    for set_no in range(8):
        np.random.seed(1)
        X_basic_set, y_set = X_sets[set_no][feature_keys_list + [se
        print("Mean Quadratic Kappa Score - Set", (set_no+1), ":",
    print()
```

```
Basic 11 features ( Model:  LinearRegression )
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 1 : 0.83882298210
3871
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 2 : 0.69121342764
33134
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 3 : 0.64381973498
69839
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 4 : 0.68879196336
81174
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 5 : 0.77772630382
94968
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 6 : 0.66256763639
7427
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 7 : 0.73026154994
72598
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 8 : 0.72414548418
00593
```

```
Basic 11 features ( Model:  SVC )
[ 0.348  mins ] Mean Quadratic Kappa Score - Set 1 : 0.79721790050
1633
[ 0.918  mins ] Mean Quadratic Kappa Score - Set 2 : 0.64116641578
02055
[ 0.13  mins ] Mean Quadratic Kappa Score - Set 3 : 0.693711699006
2464
[ 0.326  mins ] Mean Quadratic Kappa Score - Set 4 : 0.67373168088
33637
[ 0.034  mins ] Mean Quadratic Kappa Score - Set 5 : 0.78894900047
00693
[ 0.198  mins ] Mean Quadratic Kappa Score - Set 6 : 0.61165414633
37525
[ 0.899  mins ] Mean Quadratic Kappa Score - Set 7 : 0.67615479859
32719
[ 2.242  mins ] Mean Quadratic Kappa Score - Set 8 : 0.66088588413
2749

Basic 11 features ( Model:  KNeighborsClassifier )
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 1 : 0.78052149870
16494
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 2 : 0.65121247394
61205
[ 0.002  mins ] Mean Quadratic Kappa Score - Set 3 : 0.66240028912
99836
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 4 : 0.67607972680
87713
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 5 : 0.76824763382
80047
[ 0.002  mins ] Mean Quadratic Kappa Score - Set 6 : 0.64308204249
81998
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 7 : 0.62190744764
31542
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 8 : 0.52181921111
56267
```

In [21]:
```
for classifier in get_all_classifiers():
    print("All features ( Model: ", type(classifier).__name__, ")")
    for set_no in range(8):
        np.random.seed(1)
        X_set, y_set = X_sets[set_no], y_sets[set_no]
        print("Mean Quadratic Kappa Score - Set", (set_no+1), ":",
    print()
```

```
All features ( Model:  LinearRegression )
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 1 : 0.84223986299
7365
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 2 : 0.69774889353
51792
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 3 : 0.64580105239
59329
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 4 : 0.68798357618
```

```
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 4 : 0.6879535570
52089
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 5 : 0.7753302631
0033
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 6 : 0.6777655888
84742
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 7 : 0.7702965489
82455

[ 0.001  mins ] Mean Quadratic Kappa Score - Set 8 : 0.7265795258
728722

All features ( Model:  SVC )
[ 0.423  mins ] Mean Quadratic Kappa Score - Set 1 : 0.8087481771
37593
[ 0.796  mins ] Mean Quadratic Kappa Score - Set 2 : 0.6697487027
84275
[ 0.148  mins ] Mean Quadratic Kappa Score - Set 3 : 0.6873761900
77291
[ 0.294  mins ] Mean Quadratic Kappa Score - Set 4 : 0.6600822937
612518
[ 0.036  mins ] Mean Quadratic Kappa Score - Set 5 : 0.7818935765
351112
[ 0.152  mins ] Mean Quadratic Kappa Score - Set 6 : 0.6591080373
539515
[ 0.697  mins ] Mean Quadratic Kappa Score - Set 7 : 0.7319046797
812278
[ 1.544  mins ] Mean Quadratic Kappa Score - Set 8 : 0.6546936241
226258

All features ( Model:  KNeighborsClassifier )
[ 0.002  mins ] Mean Quadratic Kappa Score - Set 1 : 0.7869545642
267217
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 2 : 0.6447356498
825076
[ 0.002  mins ] Mean Quadratic Kappa Score - Set 3 : 0.6584991177
448526
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 4 : 0.6747449448
635495
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 5 : 0.7710974661
365135
[ 0.002  mins ] Mean Quadratic Kappa Score - Set 6 : 0.6621873580
092182
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 7 : 0.6476275781
396102
[ 0.001  mins ] Mean Quadratic Kappa Score - Set 8 : 0.5105334445
413325
```

In [22]:
```
# Hyper paramter tuning SVM
print("Parameter tuning for SVM - Set 1")
Cs = [0.025, 0.05, 0.1, 0.5, 0.9]
for C in Cs:
    svm = SVC(kernel="linear", C=C)
    np.random.seed(1)
    k, _ = evaluate(X_sets[0], y_sets[0], svm)
    print("C:", C, " Mean Quadratic Kappa Score:", k)
print()

print("Paramter tuning for KNN - Set 1")
NNs = [10, 15, 25, 30]
for nn in NNs:
    knn = KNeighborsClassifier(nn)
    np.random.seed(1)
    k, _ = evaluate(X_sets[0], y_sets[0], knn)
    print("NN:", nn, " Mean Quadratic Kappa Score:", k)
print()
```

```
Parameter tuning for SVM - Set 1
[ 0.395  mins ] C: 0.025  Mean Quadratic Kappa Score: 0.8087481771
337593
[ 0.694  mins ] C: 0.05  Mean Quadratic Kappa Score: 0.80849420070
49973
[ 1.197  mins ] C: 0.1  Mean Quadratic Kappa Score: 0.809140787256
4247
[ 7.595  mins ] C: 0.5  Mean Quadratic Kappa Score: 0.806739805410
095
[ 5.285  mins ] C: 0.9  Mean Quadratic Kappa Score: 0.812090953524
3104

Paramter tuning for KNN - Set 1
[ 0.002  mins ] NN: 10  Mean Quadratic Kappa Score: 0.786954564226
7217
[ 0.001  mins ] NN: 15  Mean Quadratic Kappa Score: 0.790008716942
7893
[ 0.002  mins ] NN: 25  Mean Quadratic Kappa Score: 0.795891695387
0976
[ 0.002  mins ] NN: 30  Mean Quadratic Kappa Score: 0.794538528240
7148
```

In [23]:
```
# Using all data set
# Linear Regression
print("Evaluation using basic 11 features (All Essays)")
for classifier in get_classifiers():
    np.random.seed(1)
    print("Model:", type(classifier).__name__)
    k, _ = evaluate(X_basic, y, classifier)
    print("Mean Quadratic Kappa Score:", k)
    print()
```

```
Evaluation using basic 11 features (All Essays)
Model: LinearRegression
[ 0.001  mins ] Mean Quadratic Kappa Score: 0.7470108772429251

Model: KNeighborsClassifier
[ 0.009  mins ] Mean Quadratic Kappa Score: 0.6207298517983044
```

In [24]:
```
print("Evaluation using all features (All Essays)")
for classifier in get_classifiers():
    np.random.seed(1)
    print("Model:", type(classifier).__name__)
    k, _ = evaluate(X, y, classifier)
    print("Mean Quadratic Kappa Score:", k)
    print()
```

```
Evaluation using all features (All Essays)
Model: LinearRegression
[ 0.002  mins ] Mean Quadratic Kappa Score: 0.8220726493506044

Model: KNeighborsClassifier
[ 0.011  mins ] Mean Quadratic Kappa Score: 0.6512972872681229
```

In [27]:
```
feature_kapps = []
for feature in all_feature_keys_list:
    X_feat = X_sets[0][[feature] + [sentences_key]]
    y_feat = y_sets[0]
    np.random.seed(1)
    k_feat, _ = evaluate(X_feat, y_feat)
    feature_kapps.append((feature, k_feat))

print()
feature_kapps.sort(key = lambda x:  -x[1])
for feature_kappa in feature_kapps:
    print(feature_kappa)
```

```
[ 0.001  mins ] [ 0.001  mins ] [ 0.0  mins ] [ 0.001  mins ] [ 0.
0  mins ] [ 0.0  mins ] [ 0.0  mins ] [ 0.001  mins ] [ 0.0  mins
] [ 0.0  mins ] [ 0.0  mins ] [ 0.0  mins ] [ 0.0  mins ] [ 0.0  m
ins ] [ 0.001  mins ] [ 0.0  mins ] [ 0.0  mins ] [ 0.0  mins ] [
0.0  mins ]
('word_count_root', 0.7960980410286996)
('diff_words_count', 0.7954901398458507)
('char_count', 0.7910115897794633)
('word_count', 0.7583759316924319)
('l5_word_count', 0.7553904289450902)
('l6_word_count', 0.746746155504795)
('nouns_count', 0.7376250693553479)
('l7_word_count', 0.7035481330589399)
('stopwords_count', 0.6763578000752763)
('verbs_count', 0.6603630069783022)
('adjective_count', 0.6457466423246295)
('sen_count', 0.6401569173193834)
('punctuations_count', 0.6366138500748759)
('l8_word_count', 0.6000779049092994)
('adverbs_count', 0.5320857870070199)
('avg_word_len', 0.16663238512047812)
('spelling_error_count', 0.13404770858767884)
('avg_sen_len', 0.08952776198193305)
('small_sentence_count', 0.08124754977859958)
```

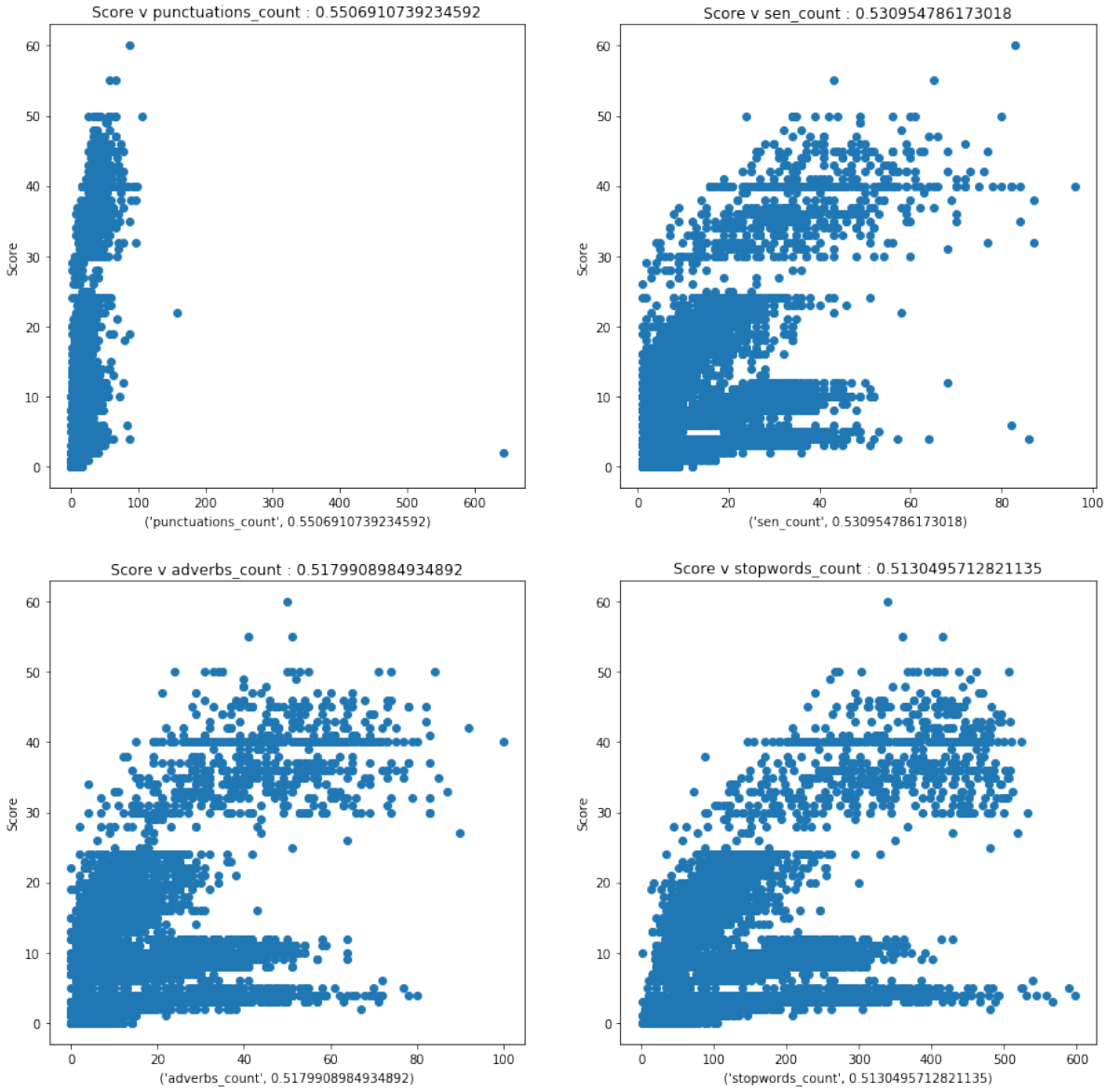
In [28]:
```
feature_kapps = []
for feature in all_feature_keys_list:
    np.random.seed(1)
    k_feat, _ = evaluate(X[[feature]+[sentences_key]], y)
    feature_kapps.append((feature, k_feat))
print()
feature_kapps.sort(key = lambda x:  -x[1])
for feature_kappa in feature_kapps:
    print(feature_kappa)
```

```
[ 0.001  mins ] [ 0.001  mins ] [ 0.001  mins ] [ 0.001  mins ] [ 0.001  mins ] [ 0.001  mins ] [ 0.001  mins ] [ 0.001  mins ] [ 0.001  mins ] [ 0.001  mins ] [ 0.001  mins ] [ 0.001  mins ] [ 0.001  mins ] [ 0.001  mins ] [ 0.001  mins ]
('punctuations_count', 0.5506910739234592)
('sen_count', 0.530954786173018)
('adverbs_count', 0.5179908984934892)
('stopwords_count', 0.5130495712821135)
('word_count', 0.5116260467573699)
('diff_words_count', 0.502868069554857)
('adjective_count', 0.470870606093578)
('char_count', 0.45132609669086265)
('word_count_root', 0.4461595319028861)
('nouns_count', 0.4456908660198316)
('l5_word_count', 0.3274018878576128)
('verbs_count', 0.2899716150221348)
('l6_word_count', 0.25525098477480274)
('avg_word_len', 0.24486383580916687)
('small_sentence_count', 0.21499266266242864)
('l7_word_count', 0.1639652888103768)
('l8_word_count', 0.06959298294167393)
('spelling_error_count', 0.013716597103376294)
('avg_sen_len', 0.00837708395438641)
```

In [29]:
```
top_features = feature_kapps[:4]

fig, ((ax1, ax2),(ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 15))
axs = [ax1, ax2, ax3, ax4]
for i in range(len(top_features)):
    ax = axs[i]
    top_feature = top_features[i]
    ax.scatter(X[[top_feature[0]]],y)
    ax.set_ylabel('Score')
    ax.set_xlabel(top_feature)
    ax.set_title("Score v " + top_feature[0] + " : " + str(top_feat

fig.savefig('./figs/ScoreVFeature.png', bbox_inches = "tight")
```



In [ ]:
```
print("Mean of features:")
X[all_features_key].mean(axis=0)
```

In [ ]:
```
print("Deviation of features:")
X.std(axis=0)
```

In [79]:
```
%matplotlib inline

print("Evaluation using all features (All Essays)")

np.random.seed(1)
classifier = LinearRegression()
print("Model:", type(classifier).__name__)
k, _ = evaluate(X, y, classifier, wordvec=True)
print("Mean Quadratic Kappa Score:", k)
print()
```

```
Evaluation using all features (All Essays)
Model: LinearRegression
[ 3.532  mins ] Mean Quadratic Kappa Score: 0.9320409111276302
```

In [85]:
```
for set_no in range(8):
    np.random.seed(1)
    X_set, y_set = X_sets[set_no], y_sets[set_no]
    classifier = LinearRegression()
    k, _ = evaluate(X_set, y_set, classifier, wordvec=True)
    print("Mean Quadratic Kappa Score - Set", (set_no+1), ":", k)
print()
```

```
[ 0.519  mins ] Mean Quadratic Kappa Score - Set 1 : 0.8080467464428379
[ 0.558  mins ] Mean Quadratic Kappa Score - Set 2 : 0.6654823125403205
[ 0.229  mins ] Mean Quadratic Kappa Score - Set 3 : 0.5891032577692663
[ 0.252  mins ] Mean Quadratic Kappa Score - Set 4 : 0.7470346365909928
[ 0.301  mins ] Mean Quadratic Kappa Score - Set 5 : 0.7717726109294422
[ 0.299  mins ] Mean Quadratic Kappa Score - Set 6 : 0.7790037430381951
[ 0.294  mins ] Mean Quadratic Kappa Score - Set 7 : 0.7499739835651755
[ 0.349  mins ] Mean Quadratic Kappa Score - Set 8 : 0.5057694155579304
```

In [*]:
```
for set_no in range(4):
    np.random.seed(1)
    X_set, y_set = X_sets[set_no+2], y_sets[set_no+2]
    classifier = LinearRegression()
    k, _ = evaluate(X_set, y_set, classifier, wordvec=True, prompt_
    print("Mean Quadratic Kappa Score - Set", (set_no+3), ":", k)
print()
```

In [ ]:
```

```