Supplemental Materials Table 1. Machine learning model Python syntax

```
# coding: utf-8

# In[ ]:


import pickle
import pandas as pd
import numpy as np
from scipy import sparse, stats

from sklearn.linear_model import Ridge
from sklearn.svm import LinearSVR
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.preprocessing import FunctionTransformer, RobustScaler
from sklearn.feature_extraction.text import TfidfVectorizer,
CountVectorizer
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.metrics import mean_squared_error
from scipy import stats

from matplotlib import pyplot as plt

from tqdm import *
import warnings

from utils import DenseTransformer


# In[ ]:


def printResults(res):
    for t_col in res:
        print("{}\t{}\t{:.2f}+/-{:.3f}".format(t_col, "mse",
np.mean([x for x in res[t_col]]), np.std([x for x in res[t_col]])))


# In[ ]:


def pad_csr(a, newshape):
    """ Pads csr_matrix with zeros. Modifies a inplace. """
    n, m = a.shape
    a._shape = newshape
    a.indptr = np.pad(a.indptr, (0, newshape[0] - n), 'edge')

def filter_nans(seq):
    """ Filters out floats (np.nan) from list """
    return np.array([x for x in seq if not isinstance(x, float)])

def pad_or_trim(seq, max_len = 250):
    """ Pads or trims seq to have max_len rows """
    n, m = seq.shape

    if n > max_len:
        seq = seq[-max_len:, :]
    elif n < max_len:
        if sparse.issparse(seq):
            pad_csr(seq, (max_len, m))
        else:
            seq = np.r_[seq, np.zeros((max_len - n, m))]
    return seq


# # Load data

# In[ ]:


data = pd.read_pickle("../data/feats.pkl")
data.head(2)


# In[ ]:


data.loc[0]


# In[ ]:


############################################################
# Drop nans from alliance
############################################################
data = data[~pd.isnull(data['current.alliance'])]
data.reset_index(inplace = True, drop = True)

Y = data['current.alliance']
print(data.shape)
print(Y.shape)


# ### Label distribution

# In[ ]:


Y.hist();
plt.show()
```

```
# In[ ]:


Y.var()


# ## Model definition

# In[ ]:


# Ridge Model
model = Ridge(random_state=42)
params = {
    'alpha':[1,0.1,0.01,0.001,0.0001,0]
}

# Stores the results
res = {}


# In[ ]:


np.random.seed(20190101)
res['random'] = []
res['avg_baseline'] = []
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    skt = KFold(n_splits = 10, random_state = 42)
    therapists = data['ucctherapistid'].unique()
    for k, (trainidx, testidx) in tqdm(enumerate(skt.split(therapists, 
therapists))):

        train, test = therapists[trainidx], therapists[testidx]
        X_train, X_test = data[data['ucctherapistid'].isin(train)], 
data[data['ucctherapistid'].isin(test)]
        Y_train, Y_test = X_train['current.alliance'], 
X_test['current.alliance']

        print(k, end = "\t")
        for x in 
set(X_train['uccclientid'].unique()).intersection(set(X_test['uccclien
tid'].unique())):

print("{}/{}".format((X_train['uccclientid'].unique().shape[0]), 
(X_test['uccclientid']).unique().shape[0]), end = ",")
        print()
```

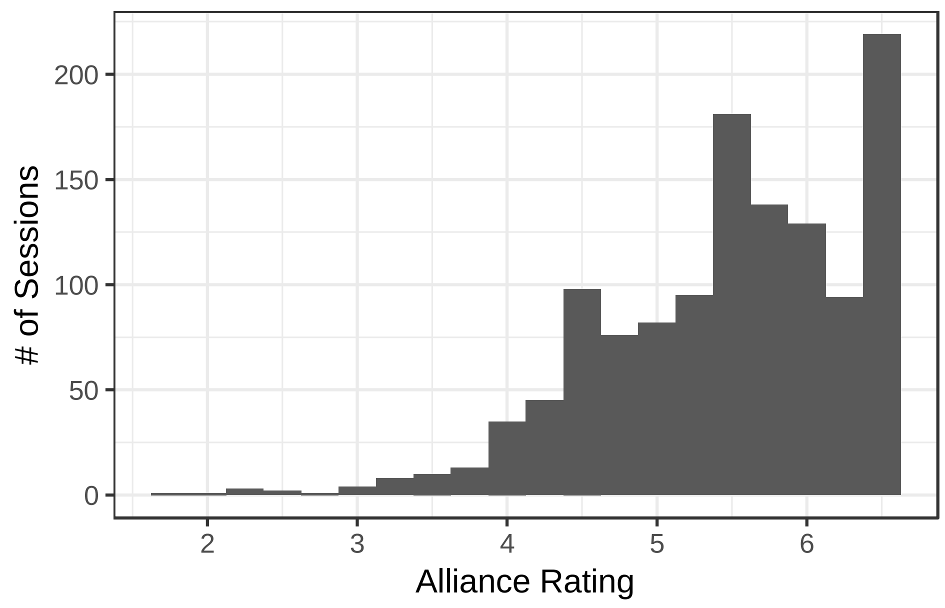
```
# ## Baselines

# In[ ]:


np.random.seed(20190101)
res['random'] = []
res['avg_baseline'] = []
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    skt = KFold(n_splits = 10, random_state = 42)
    therapists = data['ucctherapistid'].unique()
    for k, (trainidx, testidx) in tqdm(enumerate(skt.split(therapists,
therapists))):

        train, test = therapists[trainidx], therapists[testidx]
        X_train, X_test = data[data['ucctherapistid'].isin(train)],
data[data['ucctherapistid'].isin(test)]
        Y_train, Y_test = X_train['current.alliance'],
X_test['current.alliance']

        # Random baseline (with priors)
        y_pred = np.random.choice(Y_train, size=len(Y_test))

        # Score
        mse = mean_squared_error(Y_test, y_pred)

        # Append results
        res["random"].append(mse)

        # Mean
        y_pred = np.ones_like(Y_test) * Y_train.mean()

        # Score
        mse = mean_squared_error(Y_test, y_pred)

        # Append results
        res["avg_baseline"].append(mse)

printResults(res)


# ## CV estimation (N-grams)

# In[ ]:


# Bi-grams, remove English stopwords
ngram_features_T = CountVectorizer(ngram_range=(1, 2),
stop_words='english')
ngram_features_P = CountVectorizer(ngram_range=(1, 2),
stop_words='english')
```

```
# In[ ]:


np.random.seed(20190101)
res['ngrams_text_T'] = []
res['ngrams_text_P'] = []
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    for t_col in ['text_T', 'text_P']:
        skt = KFold(n_splits = 10, random_state = 42)
        therapists = data['ucctherapistid'].unique()
        for k, (trainidx, testidx) in
tqdm(enumerate(skt.split(therapists, therapists))):

            train, test = therapists[trainidx], therapists[testidx]

            X_train, X_test =
data[data['ucctherapistid'].isin(train)],
data[data['ucctherapistid'].isin(test)]

            Y_train, Y_test = X_train['current.alliance'],
X_test['current.alliance']

            xtr_t = X_train[t_col].apply(lambda x: "\n".join(x))
            xts_t = X_test[t_col].apply(lambda x: "\n".join(x))

            # Fit vocab
            xtr = ngram_features_T.fit_transform(xtr_t, Y_train)
            xts = ngram_features_T.transform(xts_t)

            # Fit model
            model.fit(xtr, Y_train)

#             # Param Search
            grid = GridSearchCV(model, params, n_jobs = 20, cv = 10,
verbose = 1)
            grid.fit(xtr, Y_train)
            model = grid.best_estimator_

            # Predict
            y_pred = model.predict(xts)

           # Score
            mse = mean_squared_error(Y_test, y_pred)

            # Append results
            res[f"ngrams_{t_col}"].append((Y_test, y_pred))

printResults(res)
```

```
# In[ ]:


x = np.hstack([np.vstack((x[0].values, x[1])) for x in
res['ngrams_text_P']])
stats.spearmanr(x[0], x[1])


# ## CV Estimation (TF-IDF)

# In[ ]:


# Bi-grams, remove English stopwords
tfidf_features_T = TfidfVectorizer(ngram_range=(1, 2),
stop_words='english', max_features = 5000)
tfidf_features_P = TfidfVectorizer(ngram_range=(1, 2),
stop_words='english', max_features = 5000)


# In[ ]:


np.random.seed(20190101)
res['tfidf_text_T'] = []
res['tfidf_text_P'] = []
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    for t_col in ['text_T', 'text_P']:
        skt = KFold(n_splits = 10, random_state = 42)
        therapists = data['ucctherapistid'].unique()
        for k, (trainidx, testidx) in
tqdm(enumerate(skt.split(therapists, therapists))):

            train, test = therapists[trainidx], therapists[testidx]

            X_train, X_test =
data[data['ucctherapistid'].isin(train)],
data[data['ucctherapistid'].isin(test)]

            Y_train, Y_test = X_train['current.alliance'],
X_test['current.alliance']

            xtr_t = X_train[t_col].apply(lambda x: "\n".join(x))
            xts_t = X_test[t_col].apply(lambda x: "\n".join(x))

            # Fit vocab
            xtr = tfidf_features_T.fit_transform(xtr_t, Y_train)
            xts = tfidf_features_T.transform(xts_t)

            # Fit model
            model.fit(xtr, Y_train)

#             # Param Search
            grid = GridSearchCV(model, params, n_jobs = 20, cv = 10,
verbose = 1)
            grid.fit(xtr, Y_train)
            model = grid.best_estimator_

             # Predict
            y_pred = model.predict(xts)

           # Score
            mse = mean_squared_error(Y_test, y_pred)

            # Append results
            res[f"tfidf_{t_col}"].append((Y_test, y_pred))

printResults(res)


# In[ ]:


x = np.hstack([np.vstack((x[0], x[1])) for x in res['s2v_text_P']])
stats.spearmanr(x[0], x[1])
mean_squared_error(x[0], x[1])


# ## CV estimation (Sent2Vec)

# In[ ]:


np.random.seed(20190101)
res['s2v_text_T'] = []
res['s2v_text_P'] = []
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    for t_col in ['text_T', 'text_P']:
        skt = KFold(n_splits = 10, random_state = 42)
        therapists = data['ucctherapistid'].unique()
        for k, (trainidx, testidx) in
tqdm(enumerate(skt.split(therapists, therapists))):

            train, test = therapists[trainidx], therapists[testidx]

            X_train, X_test =
data[data['ucctherapistid'].isin(train)],
data[data['ucctherapistid'].isin(test)]

            Y_train, Y_test = X_train['current.alliance'],
X_test['current.alliance']
```

```
            xtr = 
X_train['{}_sent2vec'.format(t_col)].apply(filter_nans).apply(pad_or_t
rim).values
            xtr = np.concatenate(xtr, axis = 0).reshape(-1, 250, 
700).sum(axis = 1)

            xts = 
X_test['{}_sent2vec'.format(t_col)].apply(filter_nans).apply(pad_or_tr
im).values
            xts = np.concatenate(xts, axis = 0).reshape(-1, 250, 
700).sum(axis = 1)

            # Fit model
            model.fit(xtr, Y_train)

            # Predict
            y_pred = model.predict(xts)

            # Score
            mse = mean_squared_error(Y_test, y_pred)

            # Append results
            res[f"s2v_{t_col}"].append(np.vstack([Y_test, y_pred]))



# In[ ]:


x = np.hstack([np.vstack((x[0], x[1])) for x in res['s2v_text_T']])
stats.spearmanr(x[0], x[1])


# ## Feature Importance
# Train ridge model on all data and then use weights to get the most 
important features

# In[ ]:


# Bi-grams, remove English stopwords
tfidf_features_T = TfidfVectorizer(ngram_range=(1, 2), 
stop_words='english', max_features = 5000)


# In[ ]:


np.random.seed(20190101)
model = Ridge(random_state=42)
```

```
with warnings.catch_warnings():
    warnings.simplefilter("ignore")

    x = data["text_T"].apply(lambda x: "\n".join(x))

    # Fit vocab
    x = tfidf_features_T.fit_transform(x, Y)

    # Fit model
    model.fit(x, Y)


# In[ ]:


k = 15
sorted_coef = np.argsort(model.coef_)
top_pos = sorted_coef[::-1][:k]
top_neg = sorted_coef[:k]

idx2word = {v:k for k, v in tfidf_features_T.vocabulary_.items()}

top_pos_words = [idx2word[x] for x in top_pos]
top_neg_words = [idx2word[x] for x in top_neg]

print(f"Top {k} most positively correlated words:
{','.join(top_pos_words)}")
print(f"Top {k} most negatively correlated words:
{','.join(top_neg_words)}")
```



Supplemental Materials Figure 1. Distribution of alliance ratings.