Team 7

Lei Liu, Yuchen He, Xiangyu Chen

Models we are using

1. BOW model
2. Word embeddings (GLOVE)
3. RNN

Experiment 1

1. Bag of word

Bag of words model: assumption for a text, ignore the word order and grammar, syntax, it just as a word set, or the term of a combination of, the emergence of each word in the text are independent, does not depend on whether the other word, or when the author of this article choose a word in an arbitrary position is not affected by the previous sentence and independent choice.

In our model, we experiment on two methods to encode our dataset.

1.

```
tokenize.fit_on_texts(train_text) # only fit on train
x_train = tokenize.texts_to_matrix(train_text, mode='count')
x_test = tokenize.texts_to_matrix(test_text, mode='count')
```

2.

```
vectorizer = CountVectorizer(min_df=0)
vectorizer.fit(train_text)
vectorizer.vocabulary_
```

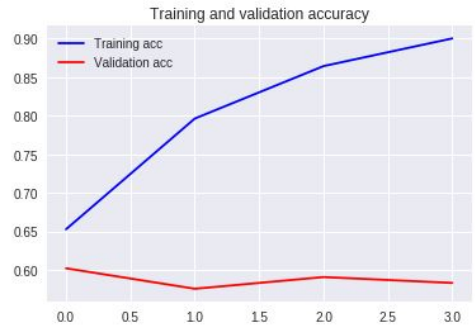
```
x_train = vectorizer.transform(train_text)
x_test = vectorizer.transform(test_text)
```

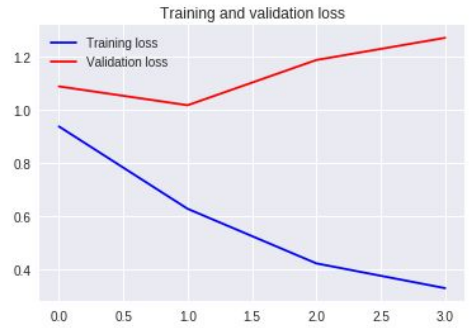
Model summary:

```
In [20]: model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               512512
_________________________________________________________________
activation_1 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 1539
_________________________________________________________________
activation_2 (Activation)    (None, 3)                 0
=================================================================
Total params: 514,051
Trainable params: 514,051
Non-trainable params: 0
_________________________________________________________________
```

Visualization:

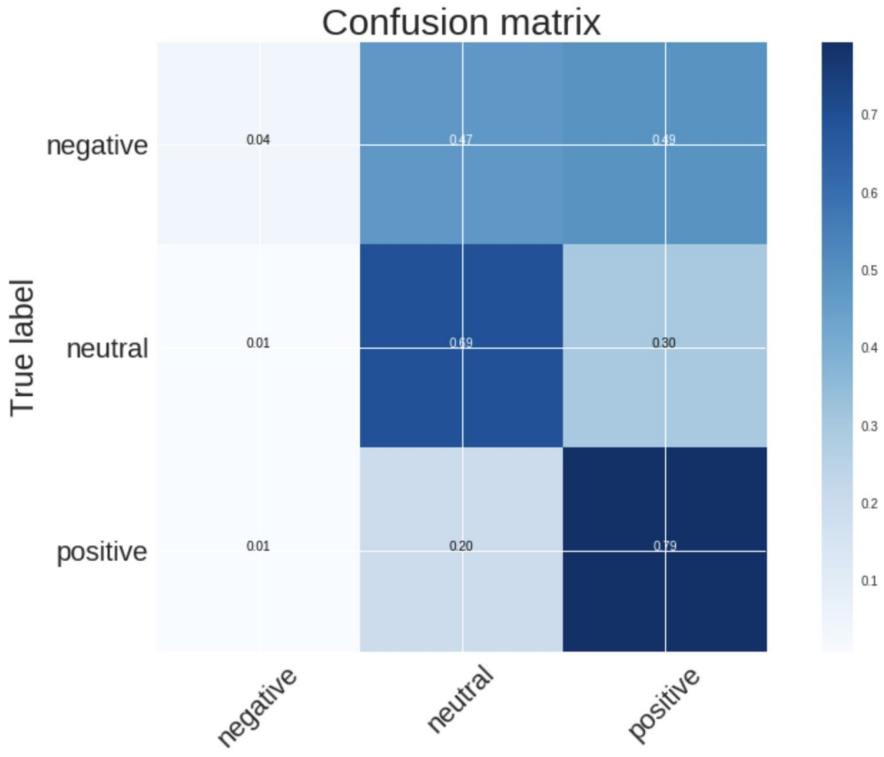




Result:

```
Test score: 1.2708252704504766
Test accuracy: 0.5833333333333334
```

Confusion Matrix:



2. Word embeddings

Model summary:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_7 (Embedding)      (None, 1000, 300)         1668900
_________________________________________________________________
flatten_7 (Flatten)          (None, 300000)            0
_________________________________________________________________
dense_12 (Dense)             (None, 128)               38400128
_________________________________________________________________
dense_13 (Dense)             (None, 3)                 387
=================================================================
Total params: 40,069,415
Trainable params: 38,400,515
Non-trainable params: 1,668,900
_________________________________________________________________
```

Visualization:

Result:

Test loss = 0.4767

Test accuracy = 0.5833

3. RNN

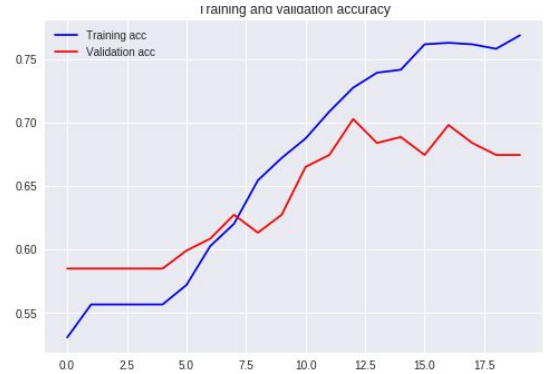
The idea behind rnn is make use of sequential information. In other words, if we want to predict a word, we want to know which words came before it. And we calculate every sequence through

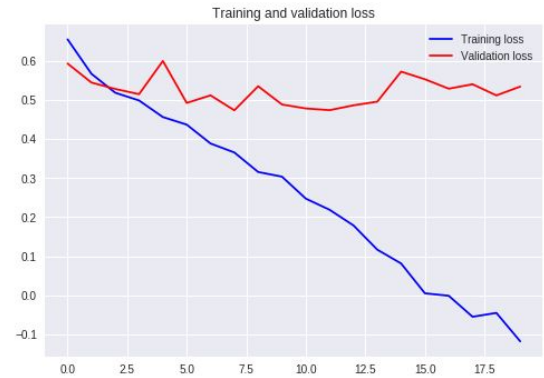
all tasks. What we use is LSTM which is a extension of RNN model itself. LSTM helps rnn remembers its inputs for quite amount of time. It is like a gate cell that decide whether or not to store or delete the data. Bidirectional rnn is a rnn that contains two hidden layers of opposite directions to the same output, in order to get more information to the network.
For modeling, we use bidirectional rnn with Long short-term memory to perform sentiment analysis.
Model summary:

```
model = Sequential()
model.add(layers.Embedding(vocabulary_size, embedding_size))
model.add(layers.Bidirectional(layers.LSTM(32)))
model.add(Dropout(0.2))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 32)          178016
_________________________________________________________________
bidirectional_1 (Bidirection (None, 64)                16640
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 194,721
Trainable params: 194,721
Non-trainable params: 0
_________________________________________________________________
```

Visualization:





Result:

```
scores = model.evaluate(X_test, Y_test, verbose=0)
print("Test accuracy", scores[1])
```

```
Test accuracy 0.6754716985630539
```

Experiment 2: Transfer learning

1. Bag of Word model

Test accuracy:

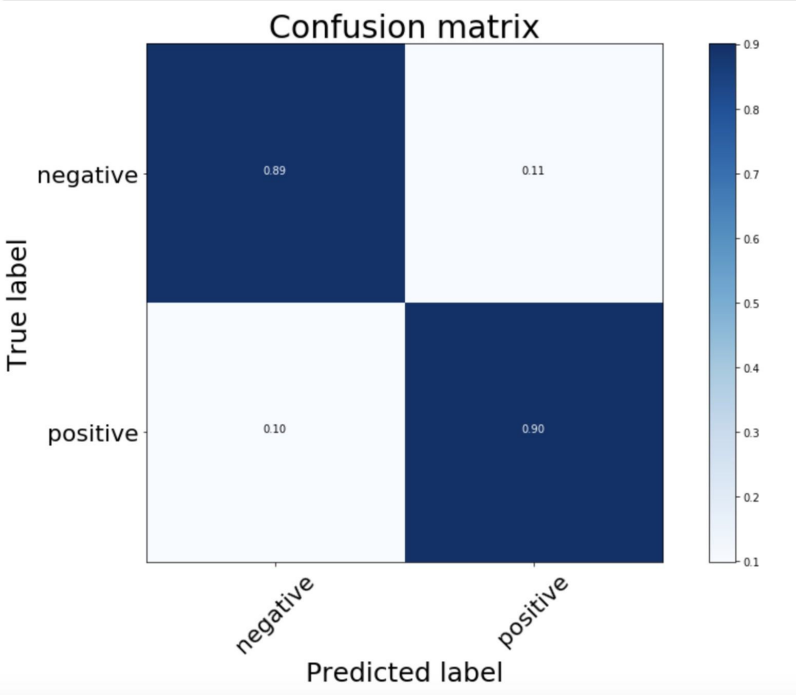
```
loss, acc = model.evaluate(x_test, y_test, verbose=0)
print('Test Accuracy: %f' % (acc*100) + '%')
```

```
Test Accuracy: 89.738000%
```

Confusion Matrix:

```
from sklearn.metrics import confusion_matrix
y_pred = model.predict(x_test)
matrix = confusion_matrix(y_test.argmax(axis=1), y_pred.argmax(axis=1))
matrix
```

```
array([[11180,  1320],
       [ 1245, 11255]])
```



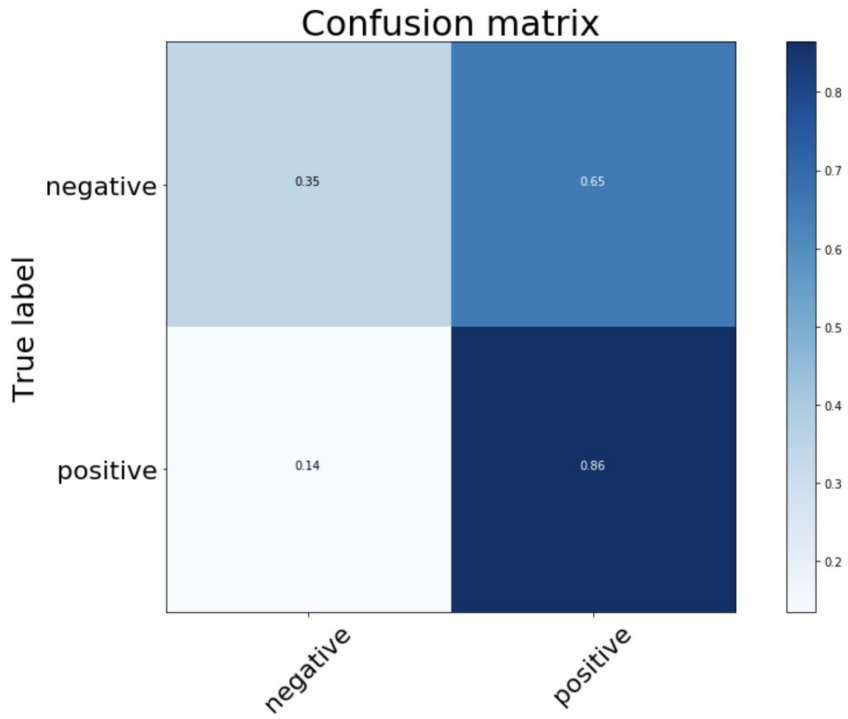
Test on financial dataset(remove neutral labels):

```
loss, acc = model.evaluate(financila_x_test, financila_y_test, verbose=0)
print('Test Accuracy on financial dataset: %f' % (acc*100) + '%')
```

```
Test Accuracy on financial dataset: 78.048780%
```

```
financila_y_pred = model.predict(financila_x_test)
financila_matrix = confusion_matrix(financila_y_test.argmax(axis=1), financila_y_pred.argmax(axis
=1))
financila_matrix
```

```
array([[ 29,  55],
       [ 61, 388]])
```



2. Word Embedding

Visualization:

Result:

```
Confusion Matrix :
[[12500     0]
 [12500     0]]
Accuracy Score : 0.5
```

3. RNN

Test accuracy:

```
scores = model.evaluate(x_test, y_test, verbose=0)
print("Test accuracy", scores[1])
```

```
Test accuracy 0.83788
```

Visualization:

Experiment 3: Using APIs

1. Google api

```
In [53]: results
```

```
    }
    sentiment {
      magnitude: 0.20000000298023224
      score: 0.20000000298023224
    }
  }, document_sentiment {
    magnitude: 0.8999999761581421
    score: 0.20000000298023224
  }
  language: "en"
  sentences {
    text {
      content: "Great."
      begin_offset: -1
    }
    sentiment {
      magnitude: 0.8999999761581421
      score: 0.8999999761581421
    }
```

```
Overall Sentiment: score of 0.10000000149011612 with magnitude of 945.4000244140625
```

2. Microsoft Azure api

```
{'documents': [{'id': '1', 'score': 0.887542188167572},
               {'id': '2', 'score': 0.832602858543396},
               {'id': '3', 'score': 0.5},
               {'id': '4', 'score': 0.9415260553359985},
               {'id': '5', 'score': 0.5},
               {'id': '6', 'score': 0.7214415073394775},
               {'id': '7', 'score': 0.5},
               {'id': '8', 'score': 0.9610069990158081},
```

```
[25] Normalized_score = sum/count
     print(Normalized_score)
```

```
0.6316885989946168
```

3. Amazon api.

```
output.head(10)
```

	{"File": "data.csv"	"Line": 0	"Sentiment": "NEUTRAL"	"SentimentScore": {"Mixed": 0.01039357390254736	"Negative": 0.058567967265844345	"Neutral": 0.8293421864509583	"Positive": 0.10169627517461777}}
0	{"File": "data.csv"	"Line": 1	"Sentiment": "POSITIVE"	"SentimentScore": {"Mixed": 0.002648639958351...	"Negative": 0.002754013054072857	"Neutral": 0.20098717510700226	"Positive": 0.793610155582428}}
1	{"File": "data.csv"	"Line": 2	"Sentiment": "NEUTRAL"	"SentimentScore": {"Mixed": 0.009224679321050644	"Negative": 0.0079100215807557 1	"Neutral": 0.793932318687439	"Positive": 0.18893304467201233}}
2	{"File": "data.csv"	"Line": 3	"Sentiment": "NEUTRAL"	"SentimentScore": {"Mixed": 0.006865730043500662	"Negative": 0.1357971727848053	"Neutral": 0.8557537794113159	"Positive": 0.0015832841163501143}}
3	{"File": "data.csv"	"Line": 4	"Sentiment": "POSITIVE"	"SentimentScore": {"Mixed": 0.01740359328687191	"Negative": 0.0007852203561924398	"Neutral": 0.2256394475698471	"Positive": 0.7561717629432678}}
4	{"File": "data.csv"	"Line": 5	"Sentiment": "NEUTRAL"	"SentimentScore": {"Mixed": 0.004719793330878019	"Negative": 0.03092263825237751	"Neutral": 0.9173388481140137	"Positive": 0.047018732875585556}}
5	{"File": "data.csv"	"Line": 6	"Sentiment": "NEUTRAL"	"SentimentScore": {"Mixed": 0.009875085204839706	"Negative": 0.07467585802078247	"Neutral": 0.8431217670440674	"Positive": 0.07232736796140671}}
6	{"File": "data.csv"	"Line": 7	"Sentiment": "NEUTRAL"	"SentimentScore": {"Mixed": 0.002160169649869...	"Negative": 0.0043444205075502396	"Neutral": 0.9272862672805786	"Positive": 0.06620916724205017}}
7	{"File": "data.csv"	"Line": 8	"Sentiment": "POSITIVE"	"SentimentScore": {"Mixed": 0.028535146266222	"Negative": 0.00031787517946213484	"Neutral": 0.013152285479009151	"Positive": 0.9579946994781494}}
8	{"File": "data.csv"	"Line": 9	"Sentiment": "NEUTRAL"	"SentimentScore": {"Mixed": 0.004719793330878019	"Negative": 0.03092263825237751	"Neutral": 0.9173388481140137	"Positive": 0.047018732875585556}}
9	{"File": "data.csv"	"Line": 10	"Sentiment": "NEUTRAL"	"SentimentScore": {"Mixed": 0.002401404781267047	"Negative": 0.01871020346879959	"Neutral": 0.9573071599006653	"Positive": 0.021581171080470085}}

4. Watson api


```
{
  "usage": {
    "text_units": 1,
    "features": 2,
    "text_characters": 271
  },
  "keywords": [
    {
      "count": 1,
      "relevance": 0.948795,
      "sentiment": {
        "label": "positive",
        "score": 0.997936
      },
      "text": "total revenues"
    },
    {
      "count": 1,
      "relevance": 0.874435,
```


```
In [167]: score

Out[167]: 0.2163279311885612
```

Average normalized score for the database s 0.31600551006.

Experiment 4: Ensemble learning using AutoML

1. TPOT

Best model by far, due to early stop

```
Best pipeline: XGBClassifier(input_matrix, learning_rate=0.01, max_depth=7, min_child_weight=15, n_estimat
ors=100, nthread=1, subsample=0.6500000000000001)
TPOTClassifier(config_dict=None, crossover_rate=0.1, cv=5,
        disable_update_check=False, early_stop=None, generations=5,
        max_eval_time_mins=5, max_time_mins=None, memory=None,
        mutation_rate=0.9, n_jobs=1, offspring_size=None,
        periodic_checkpoint_folder=None, population_size=20,
        random_state=42, scoring=None, subsample=1.0, use_dask=False,
        verbosity=2, warm_start=False)
```

Result:

```
print(pipeline_optimizer.score(X_test, Y_test))
```

```
0.864406779661017
```

This model is better than metrics from Experiment 3

2. Auto SKLearn

Best model:

```
{'balancing:strategy': 'none',
 'categorical_encoding:__choice__': 'one_hot_encoding',
 'categorical_encoding:one_hot_encoding:minimum_fraction': 0.01,
 'categorical_encoding:one_hot_encoding:use_minimum_fraction': 'True',
 'classifier:__choice__': 'random_forest',
 'classifier:random_forest:bootstrap': 'True',
 'classifier:random_forest:criterion': 'gini',
 'classifier:random_forest:max_depth': 'None',
 'classifier:random_forest:max_features': 0.5,
 'classifier:random_forest:max_leaf_nodes': 'None',
 'classifier:random_forest:min_impurity_decrease': 0.0,
 'classifier:random_forest:min_samples_leaf': 1,
 'classifier:random_forest:min_samples_split': 2,
 'classifier:random_forest:min_weight_fraction_leaf': 0.0,
 'classifier:random_forest:n_estimators': 100,
 'imputation:strategy': 'mean',
 'preprocessor:__choice__': 'no_preprocessing',
 'rescaling:__choice__': 'standardize'}
```

Result:

```
accuracy on training set: 0.800000
accuracy on test set: 0.441509
```

This model is better than metrics from Experiment 3

3. H2o. ai

```
# View the AutoML Leaderboard
lb = aml.leaderboard
lb.head(rows=lb.nrows)  # Print all rows instead of default (10 rows)
```

model_id	mean_per_class_error	logloss	rmse	mse
GBM_3_AutoML_20190320_222459	0.575257	1.00263	0.614912	0.378116
StackedEnsemble_BestOfFamily_AutoML_20190320_222459	0.576643	0.891261	0.570582	0.325564
GBM_4_AutoML_20190320_222459	0.580169	1.00139	0.614542	0.377662
StackedEnsemble_AllModels_AutoML_20190320_222459	0.580867	0.898538	0.573022	0.328355
GBM_2_AutoML_20190320_222459	0.585912	1.00351	0.615014	0.378243
DRF_1_AutoML_20190320_222459	0.586274	25.4746	0.861694	0.742517
GBM_1_AutoML_20190320_222459	0.591442	1.00806	0.616403	0.379952
XRT_1_AutoML_20190320_222459	0.600998	6.26772	0.604651	0.365602
DeepLearning_grid_1_AutoML_20190320_222459_model_1	0.610937	1.27363	0.632715	0.400328
GBM_grid_1_AutoML_20190320_222459_model_2	0.615114	1.39495	0.607832	0.36946
GBM_grid_1_AutoML_20190320_222459_model_1	0.615954	1.08246	0.634845	0.403029
GBM_5_AutoML_20190320_222459	0.644038	0.977829	0.617127	0.380846
DeepLearning_1_AutoML_20190320_222459	0.650065	1.04893	0.612698	0.375399
XGBoost_grid_1_AutoML_20190320_222459_model_2	0.662276	0.920755	0.590503	0.348694
XGBoost_1_AutoML_20190320_222459	0.666667	0.937698	0.599206	0.359048
XGBoost_3_AutoML_20190320_222459	0.666667	0.979291	0.618823	0.382942
XGBoost_grid_1_AutoML_20190320_222459_model_4	0.666667	0.942253	0.601085	0.361303
XGBoost_2_AutoML_20190320_222459	0.666667	0.971734	0.615961	0.379408
XGBoost_grid_1_AutoML_20190320_222459_model_1	0.666667	0.945237	0.601561	0.361875
XGBoost_grid_1_AutoML_20190320_222459_model_3	0.666667	0.953771	0.605054	0.36609
DeepLearning_grid_1_AutoML_20190320_222459_model_2	0.666667	1.30682	0.652082	0.425211
GLM_grid_1_AutoML_20190320_222459_model_1	0.666667	0.955539	0.597254	0.356713

Best model:

```
# The leader model is stored here
aml.leader
```

```
Model Details
=============
H2OGradientBoostingEstimator :  Gradient Boosting Machine
Model Key:  GBM_3_AutoML_20190320_222459


ModelMetricsMultinomial: gbm
** Reported on train data. **

MSE: 0.003479620781523639
RMSE: 0.05898831054983385
LogLoss: 0.04024646567686367
Mean Per-Class Error: 0.002287138211001661
Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
```

negative	neutral	positive	Error	Rate
139.0	0.0	0.0	0.0	0 / 139
1.0	630.0	1.0	0.0031646	2 / 632
0.0	2.0	539.0	0.0036969	2 / 541
140.0	632.0	540.0	0.0030488	4 / 1,312

This model is better than metrics from Experiment 3

Conclusion:

Our best model so far for auto Machine Learning is using TPO, accuracy is 0.8644