Malware Classification using LSTM (multi-class)

Author: Shamli

In [1]:
```
# basic imports
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# tensorflow import
import tensorflow as tf
print(tf.__version__)
```

```
2.1.0
```

In [3]:
```
# check if tensorflow works with CUDA
tf.test.is_built_with_cuda()
```

Out[3]:
```
True
```

In [4]:
```
tf.test.is_gpu_available(cuda_only=False, min_cuda_compute_capability=None)
```

```
WARNING:tensorflow:From <ipython-input-4-78f884b5c1a9>:1: is_gpu_available
(from tensorflow.python.framework.test_util) is deprecated and will be remo
ved in a future version.
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
```

Out[4]:
```
True
```

Train/Test splits and Model

1. Labelled features

In [14]:
```
# import all data
lf_df = pd.read_csv('labelledfeatures.csv') # labelled feature vectors
```

In [15]:
```
lf_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6300 entries, 0 to 6299
Columns: 330 entries, V1 to Filename
dtypes: int64(329), object(1)
memory usage: 15.9+ MB
```

In [22]:
```
# feature vectors split as X and Y for labelled data
X_lf = lf_df.iloc[:, :-2]
Y_lf = lf_df.iloc[:, 328]
```

In [23]:
```
# train/test split manually
X_lf_train = X_lf[0:5600]
Y_lf_train = Y_lf[0:5600]
X_lf_test = X_lf[5600:]
Y_lf_test = Y_lf[5600:]
```

In [24]:
```
# more imports
from sklearn.preprocessing import StandardScaler

# scaling
scaler = StandardScaler()
scaler.fit(X_lf_train)
X_lf_train = scaler.transform(X_lf_train)
X_lf_test = scaler.transform(X_lf_test)
```

In [25]:
```
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

# encoding for multiclass target
encode = LabelEncoder()
encode.fit(Y_lf_train)
Y_lf_train = encode.transform(Y_lf_train)
Y_lf_train = to_categorical(Y_lf_train, num_classes=7)
```

In [26]:

```
# keras import
from keras.models import Sequential
from keras.layers import LSTM, Dropout, InputLayer
from keras.layers.core import Dense

# reshape
rows = X_lf_train.shape[0]
cols = X_lf_train.shape[1]
X_lf_train = X_lf_train.reshape(rows, 1, cols)
# X_lf_train.shape

# build model
model_lf = Sequential()
model_lf.add(InputLayer(input_shape=(1, 328)))
model_lf.add(LSTM(128, activation='relu', return_sequences=False))
model_lf.add(Dense(10, activation='relu', kernel_initializer='uniform'))
model_lf.add(Dropout(0.2))
model_lf.add(Dense(7, activation='softmax'))
model_lf.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
# model_lf.compile(optimizer='adam', loss='categorical_crossentropy', metrics
print(model_lf.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 128)               233984
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290
_________________________________________________________________
dropout_3 (Dropout)          (None, 10)                0
_________________________________________________________________
dense_5 (Dense)              (None, 7)                 77
=================================================================
Total params: 235,351
Trainable params: 235,351
Non-trainable params: 0
_________________________________________________________________
None
```

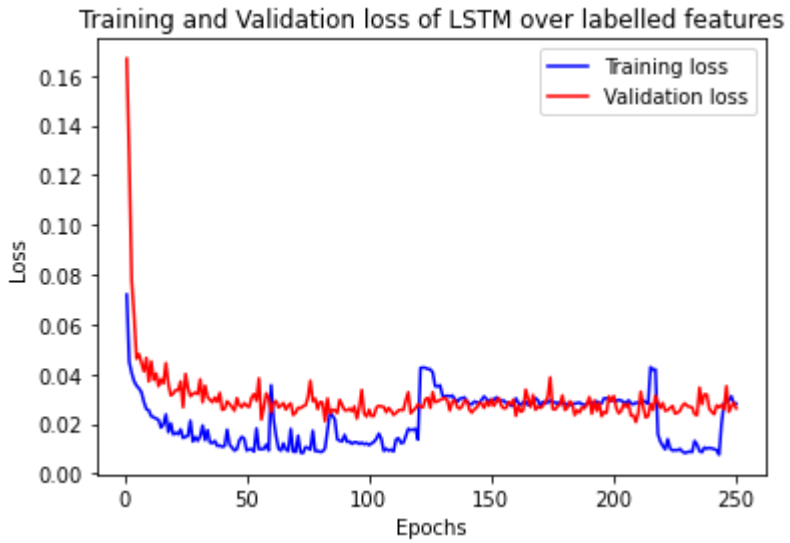
In [27]:

```
import datetime as dt

start = dt.datetime.now()
history_lf = model_lf.fit(X_lf_train, Y_lf_train, epochs=250, validation_spli
end = dt.datetime.now()
lf_traintime = end - start
print("\nTime taken for training: {}".format(lf_traintime))
```

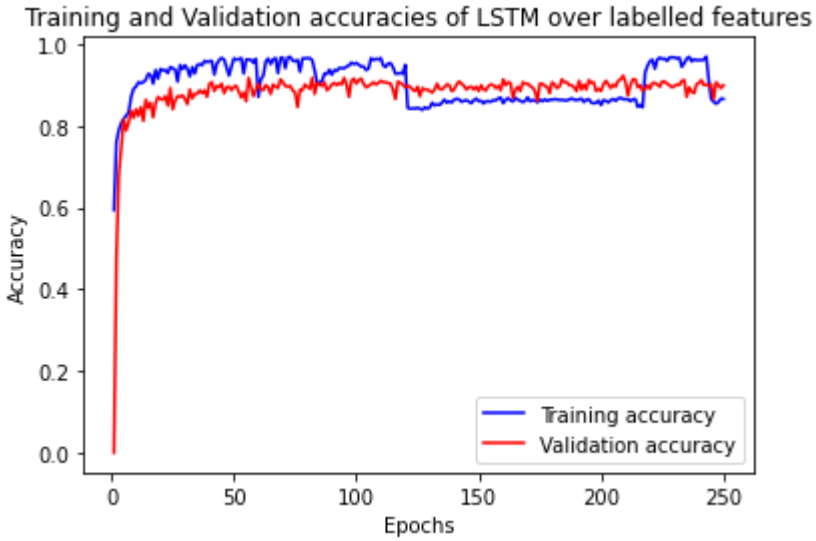
```
Epoch 232/250
5040/5040 [==============================] - 1s 297us/step - loss: 0.0087
- accuracy: 0.9675 - val_loss: 0.0266 - val_accuracy: 0.9000
Epoch 233/250
5040/5040 [==============================] - 1s 292us/step - loss: 0.0132
- accuracy: 0.9450 - val_loss: 0.0239 - val_accuracy: 0.9107
Epoch 234/250
5040/5040 [==============================] - 2s 301us/step - loss: 0.0117
- accuracy: 0.9544 - val_loss: 0.0235 - val_accuracy: 0.9125
Epoch 235/250
5040/5040 [==============================] - 2s 302us/step - loss: 0.0091
- accuracy: 0.9651 - val_loss: 0.0346 - val_accuracy: 0.8714
Epoch 236/250
5040/5040 [==============================] - 2s 301us/step - loss: 0.0086
- accuracy: 0.9694 - val_loss: 0.0279 - val_accuracy: 0.8946
Epoch 237/250
5040/5040 [==============================] - 2s 299us/step - loss: 0.0105
- accuracy: 0.9583 - val_loss: 0.0319 - val_accuracy: 0.8839
Epoch 238/250
5040/5040 [==============================] - 2s 305us/step - loss: 0.0101
```

In [28]:

```
# Loss graphs
loss = history_lf.history['loss']
val_loss = history_lf.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss of LSTM over labelled features')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [29]:
```
# accuracy graphs
acc = history_lf.history['accuracy']
val_acc = history_lf.history['val_accuracy']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
plt.title('Training and Validation accuracies of LSTM over labelled features')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [30]:
```
# reshape
rows = X_lf_test.shape[0]
cols = X_lf_test.shape[1]
# print(rows)
# print(cols)
X_lf_test = X_lf_test.reshape(rows, 1, cols)
# X_lf_test.shape

# encoding for multiclass target
encode = LabelEncoder()
encode.fit(Y_lf_test)
Y_lf_test = encode.transform(Y_lf_test)
Y_lf_test = to_categorical(Y_lf_test, num_classes=7)

start = dt.datetime.now()
result_lf = model_lf.evaluate(X_lf_test, Y_lf_test, batch_size=5)
end = dt.datetime.now()
lf_testtime = end - start
print("Loss, Accuracy = {}".format(result_lf))
print("\nTime taken for testing: {}".format(lf_testtime))
```

```
700/700 [==============================] - 1s 772us/step
Loss, Accuracy = [0.0357953757096126, 0.8357142806053162]

Time taken for testing: 0:00:00.854529
```

2. Labelled Word2Vec

In [31]:
```
lwv_df = pd.read_csv('labelledW2Vec.csv') # labelled word2vec
```

In [32]:
```
lwv_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6300 entries, 0 to 6299
Columns: 514 entries, V1 to Filename
dtypes: float64(512), int64(1), object(1)
memory usage: 24.7+ MB
```

In [33]:
```
# word2vec split as X and Y for labelled data
X_lwv = lwv_df.iloc[:, :-2]
Y_lwv = lwv_df.iloc[:, 512]
```

In [34]:
```
# train/test split manually
X_lwv_train = X_lwv[0:5600]
Y_lwv_train = Y_lwv[0:5600]
X_lwv_test = X_lwv[5600:]
Y_lwv_test = Y_lwv[5600:]
```

In [35]:

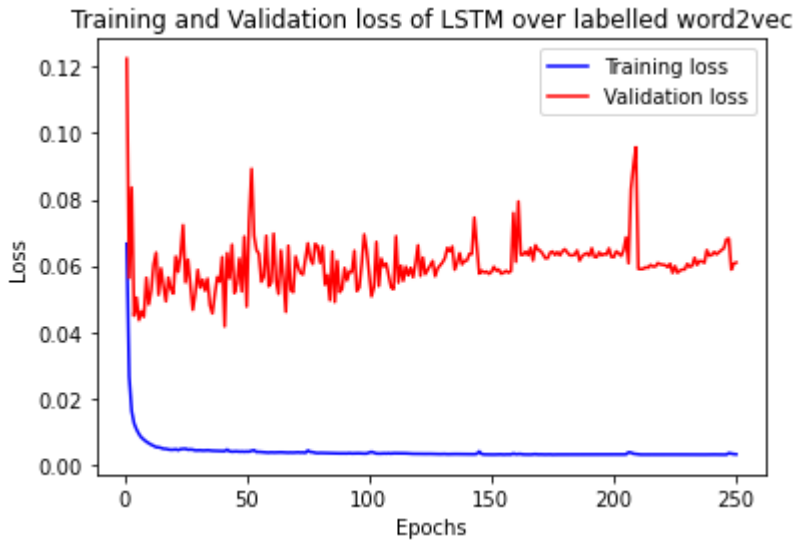
```
# reshape
rows = X_lwv_train.shape[0]
cols = X_lwv_train.shape[1]
X_lwv_train = X_lwv_train.values.reshape(rows, 1, cols)
# X_lwv_train.shape

# encoding for multiclass target
encode = LabelEncoder()
encode.fit(Y_lwv_train)
Y_lwv_train = encode.transform(Y_lwv_train)
Y_lwv_train = to_categorical(Y_lwv_train, num_classes=7)

# build model
model_lwv = Sequential()
model_lwv.add(InputLayer(input_shape=(1, 512)))
model_lwv.add(LSTM(128, activation='relu', return_sequences=False))
model_lwv.add(Dense(10, activation='relu', kernel_initializer='uniform'))
model_lf.add(Dropout(0.2))
model_lwv.add(Dense(7, activation='softmax'))
model_lwv.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
print(model_lwv.summary())
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 128)               328192
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1290
_________________________________________________________________
dense_7 (Dense)              (None, 7)                 77
=================================================================
Total params: 329,559
Trainable params: 329,559
Non-trainable params: 0
_________________________________________________________________
None
```

In [36]:
```
start = dt.datetime.now()
history_lwv = model_lwv.fit(X_lwv_train, Y_lwv_train, epochs=250, validation_
end = dt.datetime.now()
lwv_traintime = end - start
print("\nTime taken for training: {}".format(lwv_traintime))
```

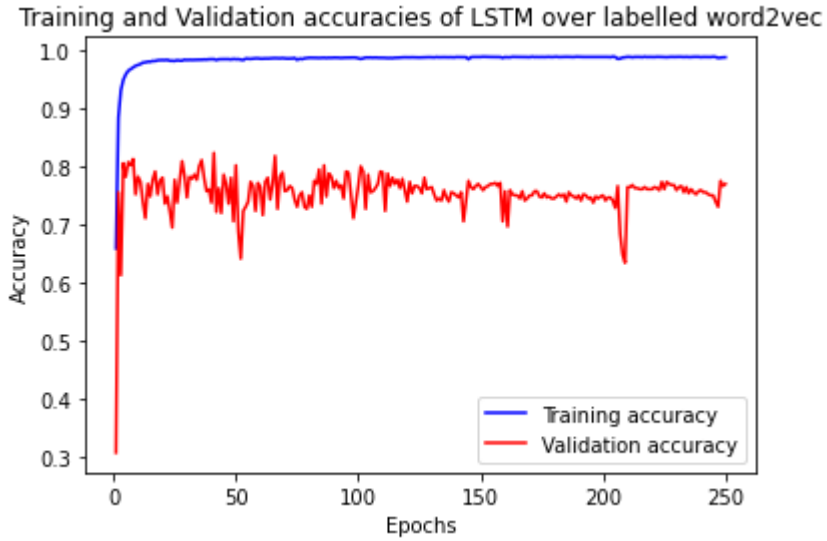
```
5040/5040 [==============================] - 1s 286us/step - loss: 0.0031
- accuracy: 0.9885 - val_loss: 0.0600 - val_accuracy: 0.7661
Epoch 233/250
5040/5040 [==============================] - 1s 295us/step - loss: 0.0031
- accuracy: 0.9881 - val_loss: 0.0624 - val_accuracy: 0.7554
Epoch 234/250
5040/5040 [==============================] - 1s 288us/step - loss: 0.0031
- accuracy: 0.9885 - val_loss: 0.0602 - val_accuracy: 0.7643
Epoch 235/250
5040/5040 [==============================] - 1s 291us/step - loss: 0.0031
- accuracy: 0.9881 - val_loss: 0.0614 - val_accuracy: 0.7536
Epoch 236/250
5040/5040 [==============================] - 2s 298us/step - loss: 0.0031
- accuracy: 0.9881 - val_loss: 0.0613 - val_accuracy: 0.7607
Epoch 237/250
5040/5040 [==============================] - 2s 299us/step - loss: 0.0031
- accuracy: 0.9885 - val_loss: 0.0609 - val_accuracy: 0.7554
Epoch 238/250
5040/5040 [==============================] - 1s 294us/step - loss: 0.0031
- accuracy: 0.9885 - val_loss: 0.0647 - val_accuracy: 0.7518
```

In [37]:
```
# Loss graphs
loss = history_lwv.history['loss']
val_loss = history_lwv.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss of LSTM over labelled word2vec')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [38]:

```
# accuracy graphs
acc = history_lwv.history['accuracy']
val_acc = history_lwv.history['val_accuracy']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
plt.title('Training and Validation accuracies of LSTM over labelled word2vec'
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [39]:

```
# reshape
rows = X_lwv_test.shape[0]
cols = X_lwv_test.shape[1]
X_lwv_test = X_lwv_test.values.reshape(rows, 1, cols)
# X_lwv_test.shape

# encoding for multiclass target
encode = LabelEncoder()
encode.fit(Y_lwv_test)
Y_lwv_test = encode.transform(Y_lwv_test)
Y_lwv_test = to_categorical(Y_lwv_test, num_classes=7)

start = dt.datetime.now()
result_lwv = model_lwv.evaluate(X_lwv_test, Y_lwv_test, batch_size=5)
end = dt.datetime.now()
lwv_testtime = end - start
print("Loss, Accuracy = {}".format(result_lwv))
print("\nTime taken for testing: {}".format(lwv_testtime))
```

```
700/700 [==============================] - 1s 836us/step
Loss, Accuracy = [0.02481497665799796, 0.9028571248054504]

Time taken for testing: 0:00:00.922841
```

Classify using LSTM model built above

In [40]:
```
uf_df = pd.read_csv('unlabelledfeatures.csv') # unlabelled feature vectors
uwv_df = pd.read_csv('unlabelledW2Vec.csv') # unlabelled word2vec
```

In [41]:
```
uf_df.info()
uwv_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 700 entries, 0 to 699
Columns: 329 entries, Filename to V328
dtypes: int64(328), object(1)
memory usage: 1.8+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 700 entries, 0 to 699
Columns: 513 entries, Filename to V512
dtypes: float64(512), object(1)
memory usage: 2.7+ MB
```

In [42]:
```
# slicing
X_uf_files = uf_df.iloc[:, 0]
X_uf_chal = uf_df.iloc[:, 1:]

X_uwv_files = uwv_df.iloc[:, 0]
X_uwv_chal = uwv_df.iloc[:, 1:]
```

1. Predict using feature vectors

In [43]:
```
# reshape
rows = X_uf_chal.shape[0]
cols = X_uf_chal.shape[1]
X_uf_chal = X_uf_chal.values.reshape(rows, 1, cols)
# X_lf_test.shape

start = dt.datetime.now()
# LSTM model for feature vectors
Y_uf_pred = model_lf.predict_classes(X_uf_chal)
end = dt.datetime.now()
uf_time = end - start
print("\nTime taken for predicting: {}".format(uf_time))
```

```
Time taken for predicting: 0:00:00.684483
```

In [44]:
```
Y_uf_pred
```

Out[44]:
```
array([4, 4, 4, 4, 6, 3, 2, 4, 1, 2, 5, 6, 3, 4, 4, 1, 3, 6, 1, 4, 1, 4,
       2, 1, 1, 4, 4, 1, 4, 5, 1, 2, 1, 1, 6, 5, 4, 4, 4, 6, 3, 6, 1, 4,
       6, 1, 6, 1, 4, 3, 6, 1, 2, 2, 4, 6, 2, 3, 1, 4, 2, 1, 4, 3, 4, 6,
       4, 5, 6, 6, 1, 4, 5, 1, 1, 6, 1, 5, 5, 5, 4, 4, 1, 4, 1, 2, 4, 1,
       4, 1, 1, 2, 2, 2, 2, 1, 2, 5, 6, 1, 6, 4, 5, 5, 4, 1, 5, 1, 1, 2,
       1, 1, 1, 1, 3, 2, 6, 3, 4, 1, 1, 2, 6, 3, 4, 5, 4, 3, 6, 2, 2, 4,
       6, 4, 4, 5, 4, 2, 6, 1, 4, 3, 2, 1, 5, 4, 6, 4, 4, 4, 4, 3, 4, 3,
       6, 6, 4, 4, 3, 2, 2, 4, 3, 1, 1, 6, 1, 4, 5, 4, 4, 4, 4, 1, 4, 3,
       2, 1, 3, 4, 1, 1, 6, 1, 2, 1, 1, 3, 3, 4, 4, 4, 1, 3, 4, 6, 1, 3,
       4, 3, 1, 6, 5, 6, 1, 2, 1, 5, 5, 1, 4, 4, 4, 2, 4, 6, 6, 5, 6, 3,
       3, 2, 1, 3, 3, 2, 4, 6, 6, 6, 1, 6, 3, 1, 3, 4, 1, 6, 4, 2, 3, 4,
       4, 6, 2, 3, 2, 1, 6, 6, 4, 2, 6, 2, 2, 2, 1, 2, 4, 1, 4, 6, 5, 3,
       6, 6, 1, 5, 3, 4, 3, 3, 1, 2, 1, 1, 6, 1, 3, 1, 2, 3, 6, 6, 4, 5,
       2, 1, 1, 3, 3, 4, 2, 4, 4, 1, 2, 4, 3, 4, 2, 6, 2, 1, 3, 5, 3, 1,
       4, 2, 2, 2, 3, 6, 2, 4, 4, 6, 3, 2, 3, 6, 1, 4, 2, 4, 4, 4, 4, 4,
       3, 4, 3, 3, 1, 5, 4, 4, 2, 5, 4, 4, 6, 5, 5, 1, 3, 6, 1, 3, 1, 6,
       2, 3, 1, 2, 4, 6, 1, 6, 2, 1, 6, 4, 6, 3, 4, 4, 4, 4, 6, 6, 1, 6,
       2, 2, 3, 3, 2, 1, 1, 6, 1, 3, 1, 2, 1, 2, 2, 5, 4, 5, 5, 4, 5, 3,
       3, 2, 6, 4, 1, 4, 3, 1, 1, 6, 6, 4, 4, 4, 4, 4, 2, 1, 4, 6, 2, 4,
```

In [51]:
```
from numpy import savetxt

savetxt('lstm_f_pred.csv', Y_uf_pred, fmt='%d')
```

2. Predict using word2vec

In [45]:
```
# reshape
rows = X_uwv_chal.shape[0]
cols = X_uwv_chal.shape[1]
X_uwv_chal = X_uwv_chal.values.reshape(rows, 1, cols)
# X_lf_test.shape

start = dt.datetime.now()
# LSTM model for feature vectors
Y_uwv_pred = model_lwv.predict_classes(X_uwv_chal)
end = dt.datetime.now()
uwv_time = end - start
print("\nTime taken for predicting: {}".format(uwv_time))
```

```
Time taken for predicting: 0:00:00.673357
```

In [46]:
```
Y_uwv_pred
```

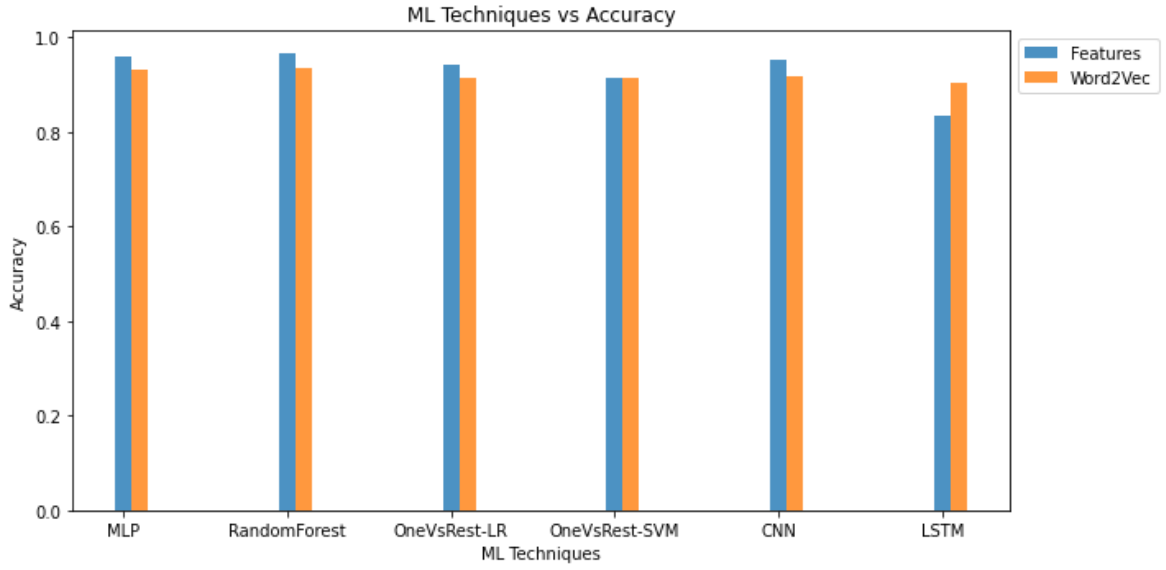
```
array([6, 6, 1, 5, 3, 0, 3, 5, 4, 6, 0, 2, 6, 1, 5, 3, 2, 5, 6, 1, 0, 5,
       2, 2, 1, 3, 3, 0, 1, 4, 4, 1, 1, 4, 3, 0, 3, 6, 6, 1, 5, 5, 3, 1,
       5, 2, 2, 2, 5, 0, 1, 0, 0, 6, 5, 2, 3, 6, 1, 0, 2, 4, 6, 4, 0, 0,
       3, 4, 3, 3, 2, 5, 0, 4, 1, 5, 4, 4, 6, 5, 5, 2, 3, 4, 1, 3, 1, 6,
       2, 3, 1, 2, 4, 6, 3, 6, 2, 3, 6, 4, 6, 3, 4, 5, 0, 4, 0, 6, 3, 6,
       2, 1, 3, 5, 2, 3, 1, 6, 1, 3, 1, 2, 1, 2, 2, 5, 4, 5, 5, 5, 5, 3,
       5, 5, 3, 0, 1, 0, 3, 0, 2, 6, 6, 4, 5, 4, 0, 0, 6, 1, 0, 6, 0, 5,
       4, 4, 3, 4, 4, 2, 2, 3, 5, 1, 1, 5, 0, 2, 1, 1, 1, 3, 1, 3, 5, 5,
       0, 6, 2, 0, 1, 2, 0, 0, 3, 2, 2, 3, 4, 0, 3, 3, 1, 2, 3, 2, 6, 0,
       6, 5, 5, 3, 5, 5, 0, 0, 4, 0, 6, 1, 0, 3, 1, 2, 0, 6, 2, 5, 5, 4,
       0, 4, 1, 5, 2, 5, 4, 4, 5, 1, 1, 6, 2, 5, 2, 2, 3, 4, 5, 4, 4, 1,
       2, 5, 6, 0, 2, 2, 0, 0, 5, 2, 4, 6, 2, 1, 2, 3, 6, 2, 6, 2, 4, 3,
       3, 2, 0, 0, 0, 3, 1, 1, 0, 6, 4, 5, 5, 6, 3, 3, 1, 4, 3, 3, 0, 2,
       5, 5, 4, 4, 6, 2, 3, 1, 0, 0, 4, 3, 2, 2, 4, 3, 1, 0, 4, 5, 2, 4,
       2, 5, 0, 0, 2, 4, 6, 3, 2, 4, 3, 4, 2, 5, 5, 2, 3, 2, 1, 3, 3, 4,
       5, 2, 3, 3, 2, 0, 1, 2, 6, 3, 0, 3, 2, 4, 2, 4, 6, 5, 0, 3, 1, 5,
       2, 3, 6, 2, 3, 6, 2, 4, 4, 4, 3, 2, 4, 4, 5, 5, 1, 1, 0, 0, 6, 5,
       3, 2, 4, 4, 5, 1, 4, 5, 2, 2, 3, 3, 6, 2, 1, 3, 5, 0, 0, 3, 2, 2,
       5, 4, 3, 2, 3, 1, 3, 0, 4, 3, 4, 4, 2, 1, 1, 2, 3, 6, 4, 4, 3, 4,
       3, 4, 1, 3, 5, 5, 6, 4, 3, 0, 5, 0, 6, 6, 0, 2, 4, 3], dtype=int6
```

In [52]:
```
savetxt('lstm_w2v_pred.csv', Y_uwv_pred, fmt='%d')
```

Graph comparison for all ML techniques used

In [66]:

```
techniques = ['MLP', 'RandomForest', 'OneVsRest-LR', 'OneVsRest-SVM', 'CNN',
features_acc = [0.96, 0.9657, 0.9428, 0.9157, 0.9542, 0.8357]
w2v_acc = [0.9314, 0.9342, 0.9157, 0.9142, 0.9171, 0.9028]

x = np.arange(len(techniques))
w = 0.1 # width
op = 0.8 # opacity

# bar plot
plt.figure(figsize=(10, 5))
barf = plt.bar(x, features_acc, w, alpha=op, label='Features')
barw = plt.bar(x+w, w2v_acc, w, alpha=op, label='Word2Vec')

plt.xlabel('ML Techniques')
plt.ylabel('Accuracy')
plt.xticks(x, techniques)
ax.set_xticklabels(techniques)
plt.title('ML Techniques vs Accuracy')
plt.legend(bbox_to_anchor=(1.0, 1.0), loc='upper left')
# plt.legend()

plt.tight_layout()
plt.show()
```



In [ ]: