This notebook is used for the project of NYU 6143 Machine Learning course project.

```
import numpy as np
import pandas as pd
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

import nltk
from nltk import word_tokenize
from collections import defaultdict
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
df = pd.read_csv("https://raw.githubusercontent.com/Kou-Guandong/6143_ML_Project/ma
```

```
counter = Counter(df['variety'].tolist())
top_10_varieties = {i[0]: idx for idx, i in enumerate(counter.most_common(10))}
df = df[df['variety'].map(lambda x: x in top_10_varieties)]
```

```
description_list = df['description'].tolist()
varietal_list = [top_10_varieties[i] for i in df['variety'].tolist()]
varietal_list = np.array(varietal_list)
```

```
count_vect = CountVectorizer()
x_train_counts = count_vect.fit_transform(description_list)
tfidf_transformer = TfidfTransformer()
                                                    (x_train_counts)
                                                    plit(x_train_tfidf, varietal_list, t
```

Saved notebook without code cell output. Notebook settings

Here it is better to use random grid search to find optimal parameters. However, to simplify the problem, we use the default parameters to show the demo.

```
clf = LogisticRegression(max_iter=1e5, tol=1e-5)
```

```
def get_model_report(clf):
```

```
    clf.fit(train_x, train_y)
    y_score = clf.predict(test_x)
    print(classification_report(test_y, y_score))
```

```
get_model_report(LogisticRegression(max_iter=1e5, tol=1e-5))
```

	precision	recall	f1-score	support
0	0.77	0.88	0.83	3945
1	0.84	0.94	0.89	3510
2	0.66	0.80	0.72	2761
3	0.84	0.78	0.81	2707
4	0.81	0.80	0.81	2083
5	0.92	0.86	0.89	1582
6	0.86	0.75	0.80	1496
7	0.81	0.60	0.69	1271
8	0.87	0.79	0.83	1045
9	0.84	0.35	0.49	997
accuracy			0.80	21397
macro avg	0.82	0.75	0.78	21397
weighted avg	0.81	0.80	0.80	21397

```
get_model_report(MultinomialNB(alpha=0.05))
```

	precision	recall	f1-score	support
0	0.72	0.82	0.76	3945
1	0.78	0.90	0.84	3510
2	0.54	0.75	0.63	2761
3	0.72	0.75	0.73	2707
4	0.70	0.71	0.71	2083
5	0.89	0.76	0.82	1582
6	0.84	0.62	0.72	1496
7	0.74	0.35	0.47	1271
8	0.84	0.71	0.77	1045
9	0.64	0.08	0.14	997
accuracy			0.72	21397
macro avg	0.74	0.65	0.66	21397
weighted avg	0.73	0.72	0.70	21397

Saved notebook without code cell output. Notebook settings

```
get_model_report(SVC(kernel='linear'))
```

	precision	recall	f1-score	support
0	0.78	0.88	0.83	3945
1	0.84	0.94	0.89	3510
2	0.65	0.81	0.72	2761
3	0.85	0.77	0.81	2707
4	0.82	0.82	0.82	2083
5	0.92	0.86	0.89	1582

```
              6       0.86      0.75      0.80      1496
              7       0.81      0.60      0.69      1271
              8       0.87      0.81      0.84      1045
              9       0.85      0.34      0.48       997

       accuracy                           0.80     21397
      macro avg       0.83      0.76      0.78     21397
   weighted avg       0.81      0.80      0.80     21397
```

```
1 get_model_report(SVC(kernel='rbf'))
```

```
              precision    recall  f1-score   support

           0       0.78      0.90      0.84      3945
           1       0.84      0.95      0.89      3510
           2       0.66      0.83      0.73      2761
           3       0.87      0.77      0.82      2707
           4       0.83      0.83      0.83      2083
           5       0.94      0.87      0.90      1582
           6       0.91      0.74      0.82      1496
           7       0.86      0.59      0.70      1271
           8       0.89      0.81      0.85      1045
           9       0.93      0.34      0.50       997

    accuracy                           0.82     21397
   macro avg       0.85      0.76      0.79     21397
weighted avg       0.83      0.82      0.81     21397
```

Neural Network

utility functions

```
1 def count_top_x_words(corpus, top_x, skip_top_n):
2     count = defaultdict(lambda: 0)
3     for c in corpus:
4         for w in word_tokenize(c):
5             count[w] += 1
6
7     return [w for w, c in sorted(count.items(), key=lambda x: x[1], r
8             if skip_top_n <= i < skip_top_n + top_x]]
9
10 def replace_top_x_words_with_vectors(corpus, top_x):
11     topx_dict = {top_x[i]: i for i in range(len(top_x))}
12     return [
13         [topx_dict[w] for w in word_tokenize(s) if w in topx_dict]
14         for s in corpus
15     ], topx_dict
16
```

Saved notebook without code cell output. Notebook settings

```
def filter_to_top_x(corpus, n_top, skip_n_top=0):
    top_x = count_top_x_words(corpus, n_top, skip_n_top)
    return replace_top_x_words_with_vectors(corpus, top_x)
```

```
from keras.models import Sequential
from keras.layers import Dense, Conv1D, Flatten
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.utils import to_categorical
```

```
mapped_list, word_list = filter_to_top_x(description_list, 2500, 10)
varietal_list_o = [top_10_varieties[i] for i in df['variety'].tolist()]
varietal_list = to_categorical(varietal_list_o)
```

```
max_review_length = 150
mapped_list = sequence.pad_sequences(mapped_list, maxlen=max_review_length)
train_x, test_x, train_y, test_y = train_test_split(mapped_list, varietal_list, tes
```

```
embedding_vector_length = 64
model = Sequential()

model.add(Embedding(2500, embedding_vector_length, input_length=max_review_length))
model.add(Conv1D(50, 5))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(max(varietal_list_o) + 1, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy
model.fit(train_x, train_y, epochs=3, batch_size=64)
```

```
Epoch 1/3
781/781 [==============================] - 36s 46ms/step - loss: 1.1174 - accura
Epoch 2/3
781/781 [==============================] - 36s 46ms/step - loss: 0.6668 - accura
Epoch 3/3
781/781 [==============================] - 36s 46ms/step - loss: 0.5629 - accura
<tensorflow.python.keras.callbacks.History at 0x7fa205ac1710>
```

Saved notebook without code cell output. Notebook settings

```
] for sc in y_score]
```

```
print(classification_report(test_y, y_score))
```

	precision	recall	f1-score	support
0	0.75	0.86	0.80	3975
1	0.81	0.91	0.86	3558
2	0.67	0.74	0.70	2784
3	0.78	0.77	0.77	2728

4	0.77	0.75	0.76	2049
5	0.89	0.82	0.86	1515
6	0.78	0.71	0.74	1539
7	0.74	0.56	0.63	1290
8	0.84	0.74	0.79	1035
9	0.71	0.33	0.45	924
micro avg	0.77	0.77	0.77	21397
macro avg	0.77	0.72	0.74	21397
weighted avg	0.77	0.77	0.76	21397
samples avg	0.77	0.77	0.77	21397

Saved notebook without code cell output. Notebook settings