**Module 18**

**Natural Language Procession (NLP)**

* [Video Transcripts](https://student.emeritus.org/courses/4765/files/3528229?wrap=1)
* [Download Video Transcripts](https://student.emeritus.org/courses/4765/files/3528229/download?download_frd=1)
* [Quick Reference Guide](https://student.emeritus.org/courses/4765/files/3528230?wrap=1)

[Module 18 Glossary](https://student.emeritus.org/courses/4765/files/3528228?wrap=1)

**Bag of Words**

A model that simplifies representations used in NLP; it represents the text as a bag of words, disregarding grammar and even word order while retaining its multiplicity

**Corpus**

A collection of documents

**Lemmatization**

A process that analyzes words and returns their base forms

**Naive Bayes**

A technique for constructing classifiers

**NLTK**

[Natural Language Toolkit](https://www.nltk.org/)—a leading platform for building Python programs to work with human language data

**Stemming**

A word analysis technique that removes the suffix of a word to derive a base word

**Stop Words**

Words that are filtered out of the results because they bring no meaning (such as "and")

**TF–IDF**

Term frequency-inverse document frequency indicates the importance of a particular word to a document in a collection.

**Token**

A piece of text, such as a word, character, or subword

**Vectorization**

A feature extraction step that obtains distinct features from the text for model training by converting the text into numerical vectors

**Install NLTK:**

Make sure “admin” on Mac!

pip install nltk

pip install -U —trusted-host pypi.org --trusted-host files.pythonhosted.org nltk

**Install wordcloud**

pip install -U —trusted-host pypi.org --trusted-host files.pythonhosted.org wordcloud

**Savio’s Session**

Import spacy

RASA

**Notes:**

There are two subfields of NLP: natural language understanding (NLU) and natural language generation (NLG). These two subfields are described as follows:

* **NLU** refers to analyzing a sentence's meaning based on the syntactic and semantic elements of the sentence
* **NLG** is the process of creating a human language text response from data input. It can also convert the text into a voice format using a text-to-speech service.

The various text preprocessing steps are:

1. Tokenization
2. Lower casing
3. Stop words removal
4. Stemming
5. Lemmatization

Text normalization includes:

* Converting all letters to lowercase or uppercase
* Converting numbers into words or removing numbers
* Removing punctuations, accent marks, and other special characters
* Removing white space
* Expanding abbreviations
* Removing stop words, sparse terms, and particular words
* Canonicalizing text

The **bag-of-words** **model**simplifies representations used in NLP. This model represents the text as a bag of words, disregarding grammar and even word order while retaining its multiplicity.

**TF–IDF** indicates the importance of a particular word to a document in a collection.

**Naive Bayes**

**Named-entity recognition (NER)**

NER is a machine learning task that uses unstructured data to extract entities, including people, places, objects, monetary values, brands, and locations. NER restricts machine learning tasks, such as text analysis or sentiment analysis, to the entities assigned as important. Thus, each industry domain must have its own NER capability to ensure maximum precision.

**Semantic search**

Semantic search is a part of NLP that uses machine learning to understand the intent behind a query, search data for the answer, and then return a response. The unique feature of semantic search is that intent is not dependent on keywords. Instead, the algorithm uses the user's search history, past purchases, online behavior, location, and other details to identify what the user is looking for and provide the most relevant information. An algorithm with a more extensive knowledge graph offers more accurate results.

**Sentiment analysis**

Sentiment analysis is one of the most widely used NLP applications in business for consumer and employee insights. NLP tasks analyze data to associate the sentiment with parts of the data categorized as entities, topics, or aspects. NLP in video content analysis studies the video and audio data and processes it like text formats. When analyzing hundreds of texts for sentiment, such as review comments or social media posts, the review subject, such as a hotel, restaurant, or movie, receives an aggregate positive, negative, or neutral score. This NLP task is widespread in social media sentiment analysis.

**Text summarizations**

As an advanced NLP technique, text summarizations summarize large documents in industries such as aerospace repair and maintenance manuals, medical journals, and research agencies. The NLP algorithms create a comprehensive dictionary of commonly occurring words. Then, they sort and categorize that dictionary. To summarize a lengthy document, the algorithms check each sentence for the words that appear most frequently in the text, then select and aggregate them.

**Speech recognition**

A speech recognition algorithm is an NLP technology that converts speech into text. A speech recognition application may also perform captioning or transcription in real time. It has applications in lectures, conferences, live interviews, news broadcasts for the hearing impaired, and video streaming services like Netflix or Hulu. Additionally, the technology can search video applications with voice commands.

**Aspect-based granularity**

Advanced NLP techniques can identify the relevant entities from the gathered data for sentiment analysis and extract the relevant information. Sentiment polarity can analyze these aspects for user emotions. The aspect-based sentiment analysis allows for highly granular sentiment analysis of data, such as customer reviews, social media posts, comments, news items, customer service emails, and chatbot data.

**Question answering system**

The question answering system is an NLP algorithm that extracts information from data, such as text documents, video data, call logs, online search history, and image repositories to answer user queries. With NLP technology, one can extract useful information from big data, so the user receives the most accurate information. Chatbot systems frequently employ this technology for customer service to improve customer satisfaction.

**Machine translations**

A machine translation technique is perhaps the most commonly used advanced NLP technique. Google and other search engines, including phones, use this application to translate millions of words every day. Machine translation played a crucial role in bringing the world closer by allowing people to understand texts written in languages unfamiliar to them. Whether it is translating recipes, music lyrics, or user manuals, machine translation helps people decode the information they would not otherwise understand. This NLP technology is also present in gadgets that translate speech automatically into the language the user specifies.

Depending on the NLP task, the evaluation metric is essential in measuring the model's performance. When assessing the quality of models in production, you would use business metrics. You can divide evaluation metrics into two categories:

* **Intrinsic evaluation** focuses on intermediate objectives (e.g., the performance of an NLP component on a specific subtask)
* **Extrinsic evaluation** is a review of the performance of the final objective (the component's performance concerning the entire application)

Extrinsic evaluation is critical since stakeholders want to know how well the model solves the business problem. However, the AI team needs to measure its performance using intrinsic evaluation metrics. Here are some intrinsic evaluation metrics that you might consider as a part of your efforts:

* Confusion matrix
* Root Mean Squared Error (RMSE)
* F1 score
* Area under the curve (AUC)
* Perplexity
* Metric for Evaluation of Translation with Explicit Ordering (METEOR)
* Recall-Oriented Understudy for Gisting Evaluation (ROUGE)

**Module Issues:**

Codio 18.2 Problem 4 & 6: what is asked not clear.

**Quizes:**

NLP aims to make computer language accessible to human beings. : False

*You are correct! The answer “*False*” is correct because NLP aims to make human language accessible to computers.*

What occurs during the preprocessing step when solving a number-based machine learning problem? : Correcting flaws in the data

*You are correct! The answer “*Correcting flaws in the data*” is correct because preprocessing corrects flaws in the data.*

What is the name of a raw dataset in NLP? : corpus

*You are correct! The answer “*corpus*” is correct because this is a raw dataset in NLP.*

Text tokenization splits the text into separate grammatical units called tokens. : True

*You are correct! The answer “*True*” is correct because tokenization is a way of separating a piece of text into smaller units called tokens.*

Which of the following are parts of the feature extraction step in NLP? *Check all that apply. :* TF-IDF, Bag-of-words model

*You are correct! The answers “*Bag-of-words model*” and “*TF-IDF*” are correct because these are part of the feature extraction step in NLP.*

What is the Python library used in NLP called? : NLTK

*You are correct! The answer “*NLTK*” is correct because this is the Python library used for NLP.*

What is the function in the NLTK Python library to tokenize a corpus? : word\_tokenize()

*You are correct! The answer “*word\_tokenize()*” is correct because this is the correct function in NLTK for tokenizing the text data.*

Consider the following text data: “You are correct.”

Which of the following is not the token formed after applying the function word\_tokenize()? : You are

*You are correct! The answer “*You are*” is correct because this is not the token formed after applying the function*word\_tokenize()*to the text data.*

Normalization operations take tokens as input and produce numbers as output. : False

*You are correct! The answer “*False*” is correct because normalization operations take tokens as input and produce tokens as output.*

Which of the following is not a stop word? : natural

*You are correct! The answer “*natural*” is correct because this is not a stop word.*

The function pos\_tag(words) in NLTK gets parts of speech into a list of words. : True

*You are correct! The answer “*True*” is correct because part-of-speech tagging in NLTK marks up the words in text format for a particular part of speech based on its definition and context.*

Stemming is the process of reducing a word to its word stem. With that in mind, what would the output be if you applied stemming logic to the following list of words?

{joy,joyful,joyfully,joyous} : joy

*You are correct! The answer “*joy*” is correct because stemming produces root forms by making a series of substring replacements.*

What is the lemmatizer used in NLTK? : WordNetLemmatizer()

*You are correct! The answer “*WordNetLemmatizer()*” is correct because this is the lemmatizer used in NLTK.*

In the bag-of-words feature extraction technique, if there are five unique tokens in the data containing ten documents, how many features would there be in the feature matrix? : Five

*You are correct! The answer “*five*” is correct because, in the bag-of-words technique, the number of features in the feature matrix is equal to the number of unique tokens in the data.*

Bag of words keeps track of informative words. : False

*You are correct! The answer “*False*” is correct because bag of words does not track how informative the words are.*

What is the correct representation of the TF–IDF equation? :tfidf(t,d) = tf(t,d) x idf(t)

*You are correct! The answer “*tfidf(t,d) = tf(t,d) × idf(t)*” is correct because the TF–IDF equation is the product of term frequency, “tf”, and inverse document frequency, “idf”.*

Consider a document containing tokens: {‘enjoy’,’disappoint’,’great’,’bore’,’bore’}.

What would the tf score for the feature “bore” be? : 0.4

*You are correct! The answer “*0.4*” is correct because the term frequency, “tf”, is equal to the number of times the term occurs in the document divided by the total number of words in that document, which comes out to 2 / 5, or 0.4.*

The formula to calculate the inverse document frequency, “idf”, is:

idf(t) = −log(number of documents that contain t / total number of documents) : True

*You are correct! The answer “*True*” is correct because the formula to calculate “idf” for a term is the negative log of the number of documents containing “t” divided by the total number of documents.*

Consider the given Python function:

train\_test\_split(test\_size=0.25)

What would be the size of the test data, provided the size of the dataset is 1,000? : 250

*You are correct! The answer “*250*” is correct because the function has the constructor*test\_size*set to 0.25, which states the test data size will be 25% of the whole dataset.*

The tokenizer provided by NLTK for tweets is Tweet\_Tokenizer. : False

*You are correct! The answer*“False”*is correct because the tokenizer provided by NLTK for the tweets is*TweetTokenizer*.*

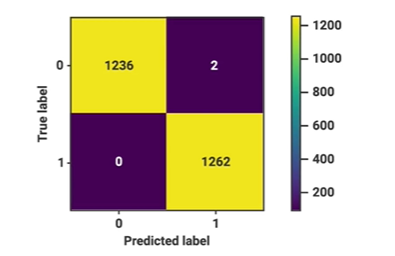
Which of the following techniques are part of preprocessing text data? *Check all that apply. :* StopWords, Stemmer, Tokenizer

*You are correct! The answers “*Tokenizer*”, “*Stemmer*”, and “*StopWords*” are correct because these techniques are part of preprocessing text data.*

NLP applies preprocessing and feature extraction on both training data and test data. : True

*You are correct! The answer*“True”*is correct because NLP applies preprocessing and feature extraction on both training data and test data for scoring.*

Consider the score of the classification model shown in the below confusion matrix:



How many mistakes did the model make? : 2

*You are correct! The answer “*2*” is correct because the confusion matrix shows the count of false negative and false positive, which are the classification model's mistakes.*

**Try-It Activity 18.1: Comparing Methods - Section B**

1. **Text preprocessing**: Consider both the CountVectorizer and TfidfVectorizer to prepare the text data for your estimator. Prepare options for stop words and max features. Also, consider both stemming and lemmatizing to normalize your text before classifying.
2. **Classification**: Consider LogisticRegression, DecisionTreeClassifier, and MultinomialNB as classification algorithms for the data. Compare their performance in terms of accuracy and speed.

**TODO:**

Check intrinsic and extrinsic evaluation

**When to Use ROC vs. Precision-Recall Curves?**

* ROC curves should be used when there are roughly equal numbers of observations for each class.
* Precision-Recall curves should be used when there is a moderate to large class imbalance.

**Data Cleaning and Preprocessing**

Duplicates - none! Dataset is balanced!

• Expanding contractions: We replaced all contractions with the expanded version of the expressions. For example, ”is not” instead of “isn’t”.

• Cleaning punctuation marks: We separated the punctuation marks6 from words to achieve cleaner sentences. For example, the sentence “This is’ (fun).” is converted to “This is ‘ ( fun ) .”

• Cleaning special characters: We replaced some special characters with an alias. For example, “alpha” instead of “α”.

**F1** = 2 \* (precision \* recall) / (precision + recall)

My results regarding

ColBERT paper used **F1** score to achieve 98.2% in their model, I used F1 score too in my results for apple-to-apple comparison.

**Visualize Word Frequency**

I used wordcloud library to visualize commonly used words as below:

plt.subplots(figsize=(16,10))

wc = wordcloud.WordCloud(stopwords=wordcloud.STOPWORDS, max\_font\_size=80, max\_words=2000,

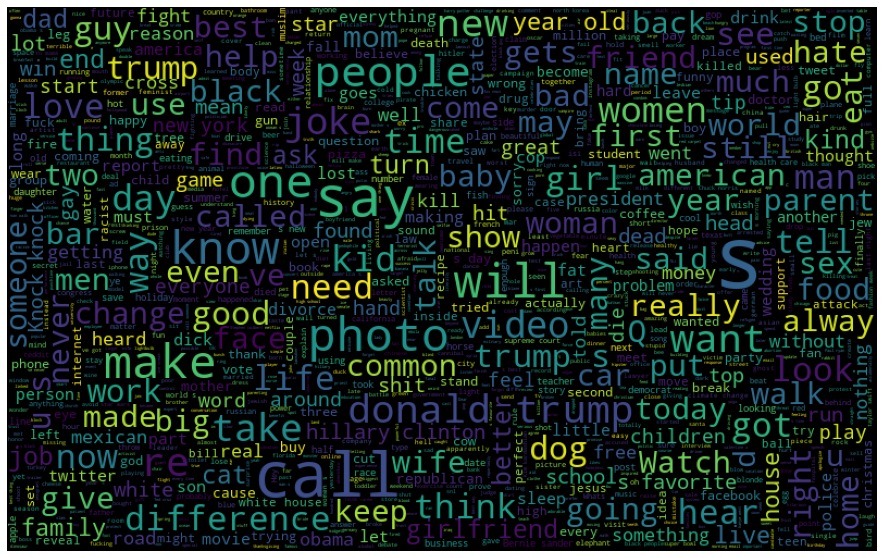
width = 800, height = 500,

background\_color='black').generate(' '.join(t for t in df['text']))

plt.axis('off')

plt.imshow(wc)

plt.show()



Computed Top 10 Words for visualizing:

%%time

# Word frequency

stop = stopwords.words('english')

freq = df['text'].str.lower().str.split(expand=True).stack().value\_counts().rename\_axis('word'). \

reset\_index(name='frequency')

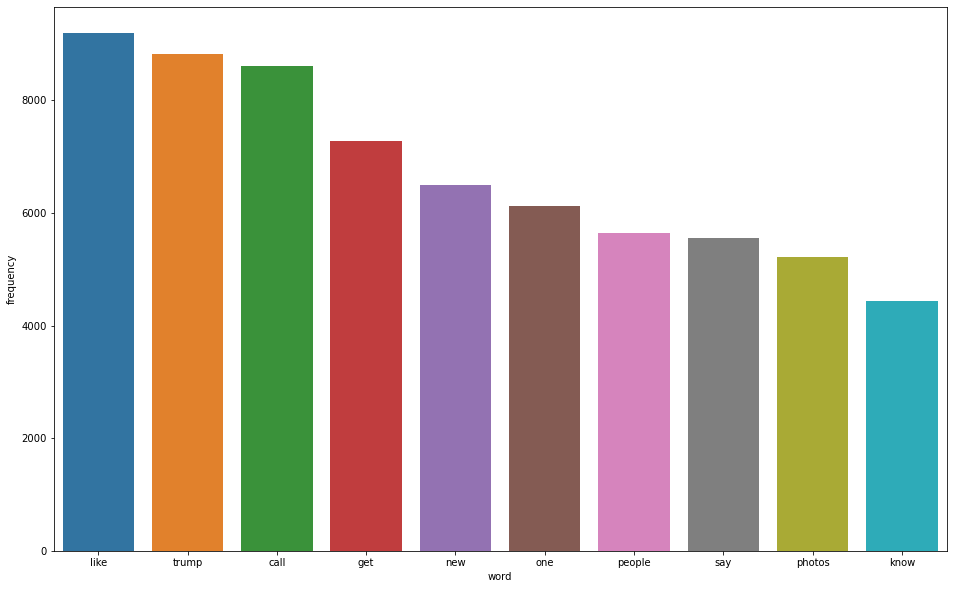
freq = freq[~freq['word'].str.lower().isin(stop)]

# Top 10 words

plt.subplots(figsize=(16,10))

sns.barplot(data=freq[0:10], x='word', y='frequency')

plt.show()



**Observations:**

There are no duplicates in the dataset. Dataset is balanced!

Each 1000 results in 1% increment in score, after 2000, each 1000 adds 0.05 to the score.

TfidfVectorizer() 25% faster than **CountVectorizer()**, however, TfidfVectorizer accuracy is slightly ~0.3% **worse**. Also, slightly better results without stop words on vectorizers.

I tried out stemmer and lemmatizer to see which vectorizer method is better; the score is ~0.02% better with **stemmed** dataset!

**Preprocessing Data and Data Cleaning**

Converting *n’t* auxiliary and *‘ll* modal verbs to *not* and *will* respectively and changing *word selection* regex to '(?u)\b[a-zA-Z]{2,}\b' in vectorizers helped ~0.2% improvement in scores.

I also removed any non-alphanumeric characters, beyond that, there is no other manipulation, no missing data:

# convert "n't" to " not"

# Remove any punctuation, special characters, numbers, anything is not alpanumerical. Replace those with space.

df = df.replace({ 'text':{"won't":'will not', "can't":'can not', "ain't":'have not', "n't":' not', "'ll":" will",

"'m":' am', '"':'', r'[^a-zA-z0-9]': ' '} }, regex=True).copy()

Empirically, I decided to go with **2500** features, **stem** and **CountVectorizer** for this dataset on all models, test/train data split is 25%/75%. No stop words option on vectorizers.

**Results**

fig, ax = plt.subplots(figsize=(10,8))

RocCurveDisplay.from\_estimator(lgr\_grid, X\_test, y\_test, pos\_label = 1, ax = ax, label = 'Logistic Regression (AUC = 0.97)')

RocCurveDisplay.from\_estimator(dtr\_grid, X\_test, y\_test, pos\_label = 1, ax = ax, label = 'Decision Tree (AUC = 0.93)')

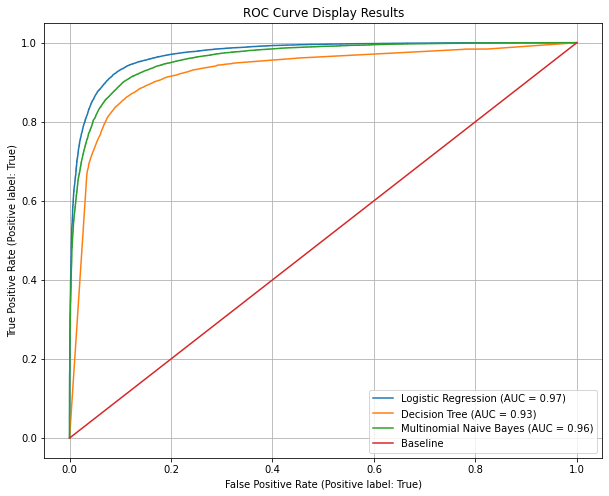
RocCurveDisplay.from\_estimator(mnb\_grid, X\_test, y\_test, pos\_label = 1, ax = ax, label = 'Multinomial Naive Bayes (AUC = 0.96)')

plt.plot(np.arange(0, 1.1, .1), np.arange(0, 1.1, .1), label = 'Baseline')

plt.title('ROC Curve Display Results')

plt.legend()

plt.grid(True)



ROC Plot shows Logistic Regression has the best AUC followed by Multinomial Naive Bayes as well as in the confusion matrix comparison below:

fig, ax = plt.subplots(1, 3, figsize = (16, 3))

ConfusionMatrixDisplay.from\_predictions(y\_test, lgr\_grid.predict(X\_test), display\_labels = ['No', 'Humorous'], ax = ax[0])

ConfusionMatrixDisplay.from\_predictions(y\_test, dtr\_grid.predict(X\_test), display\_labels = ['No', 'Humorous'], ax = ax[1])

ConfusionMatrixDisplay.from\_predictions(y\_test, mnb\_grid.predict(X\_test), display\_labels = ['No', 'Humorous'], ax = ax[2])

ax[0].grid(False)

ax[1].grid(False)

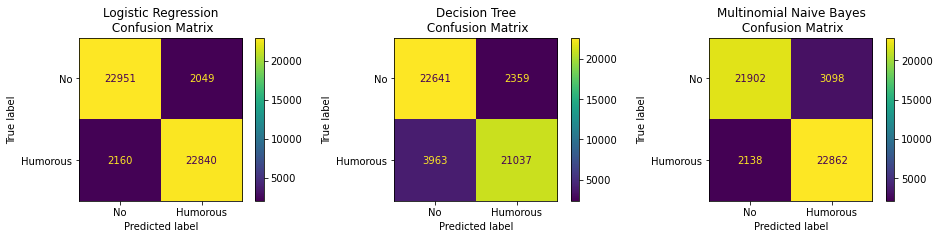
ax[2].grid(False)

ax[0].set\_title('Logistic Regression\n Confusion Matrix')

ax[1].set\_title('Decision Tree\n Confusion Matrix')

ax[2].set\_title('Multinomial Naive Bayes\n Confusion Matrix')

plt.show()



When F1 score and elapsed times compared *Logistic Regression* again is leading in **F1** score and *Multinomial Naive Bayes* is **fastest** model.

grid\_options=['Logistic Regression','Decision Tree','Multinomial Naive Bayes']

train\_accs = [lgr\_train, dtr\_train, mnb\_train]

test\_accs = [lgr\_test, dtr\_test, mnb\_test]

test\_f1s = [f1\_score(y\_test, lgr\_grid.predict(X\_test)),

f1\_score(y\_test, dtr\_grid.predict(X\_test)),

f1\_score(y\_test, mnb\_grid.predict(X\_test))]

elapsed\_times = [lgr\_time\*5, dtr\_time\*5, mnb\_time\*5]

# plot accuracy and time elapsed

fig, ax = plt.subplots(1, 2, figsize = (16, 8))

ax[0].plot(grid\_options, train\_accs, '--o', label = 'Training Accuracy')

ax[0].plot(grid\_options, test\_accs, '--o', label = 'Testing Accuracy')

ax[0].plot(grid\_options, test\_f1s, '-o', label = 'F1 Score')

ax[0].plot(np.argmax(test\_f1s), max(test\_f1s), 'ro', markersize = 12, alpha = 0.4, label = 'Best Score')

ax[0].tick\_params(axis='x', rotation=90)

ax[0].set\_xlabel('Grid Search Options')

ax[0].set\_ylabel('Accuracy Score')

ax[0].set\_title(f'Grid Search Options vs. F1 Score {grid\_options[np.argmax(test\_f1s)]} is best')

ax[0].legend()

ax[0].grid(True)

# time plot

ax[1].plot(grid\_options, elapsed\_times, '-o', label = 'Elapsed Time')

ax[1].tick\_params(axis='x', rotation=90)

ax[1].set\_xlabel('Grid Search Options')

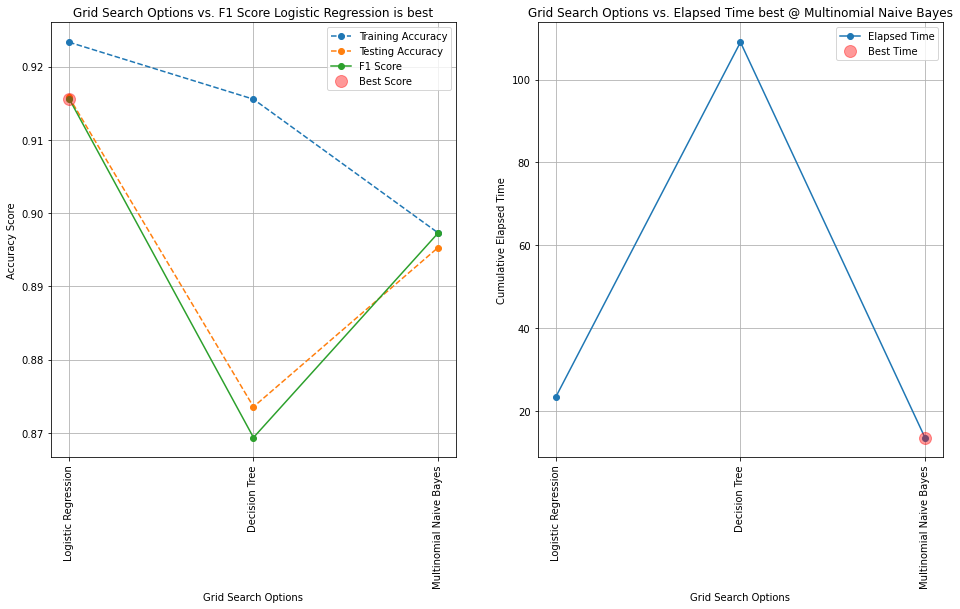
ax[1].set\_ylabel('Cumulative Elapsed Time')

ax[1].set\_title(f'Grid Search Options vs. Elapsed Time best @ {grid\_options[np.argmin(elapsed\_times)]}')

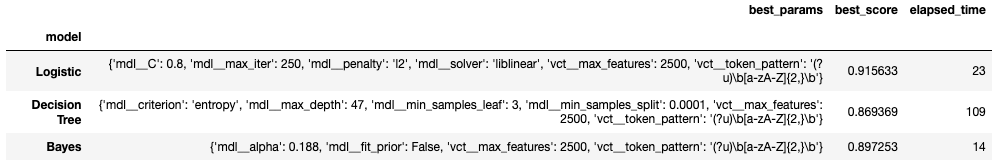
ax[1].plot(np.argmin(elapsed\_times), min(elapsed\_times), 'ro', markersize = 12, alpha = 0.4, label = 'Best Time')

ax[1].legend()

ax[1].grid(True)



**Result Summary Table**



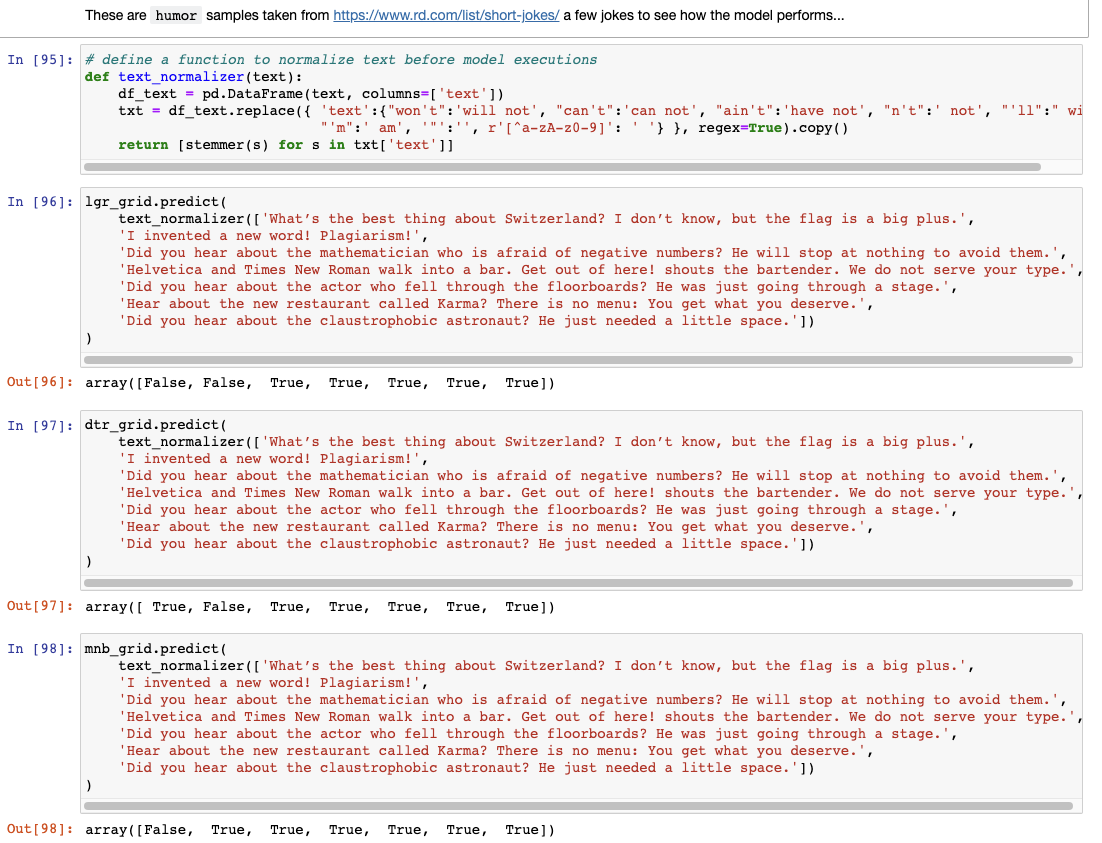
**Conclusion**

Logistic Regression seems best performer per F1 scores. Multinomial Naive Bayes is the fastest of all, Decision Tree took longer time to train and execute and its F1 is the worst.

However, I picked 7 random never seen humors at <https://www.rd.com/list/short-jokes/> and 7 random never seen ordinary sentences at <https://englishgrammarpdf.com/30-examples-of-complex-sentences-in-english-pdf/>. All models misclassified joke as non-humor, Logistic Regression:2, Decision Tree:1, and Multinomial Naive Bayes:1! On the ordinary sentences, Logistic Regression thought 6 out 7 were humors, Multinomial Naive Bayes 5 out of 7 and surprisingly Decision Tree misclassified only 3 ordinary sentences! It is not a comprehensive set to conclude anything but Decision Tree outperformed consistently even though its score is the lowest of all!

**Supplemental Information Regarding Manual Runs**

Humor:

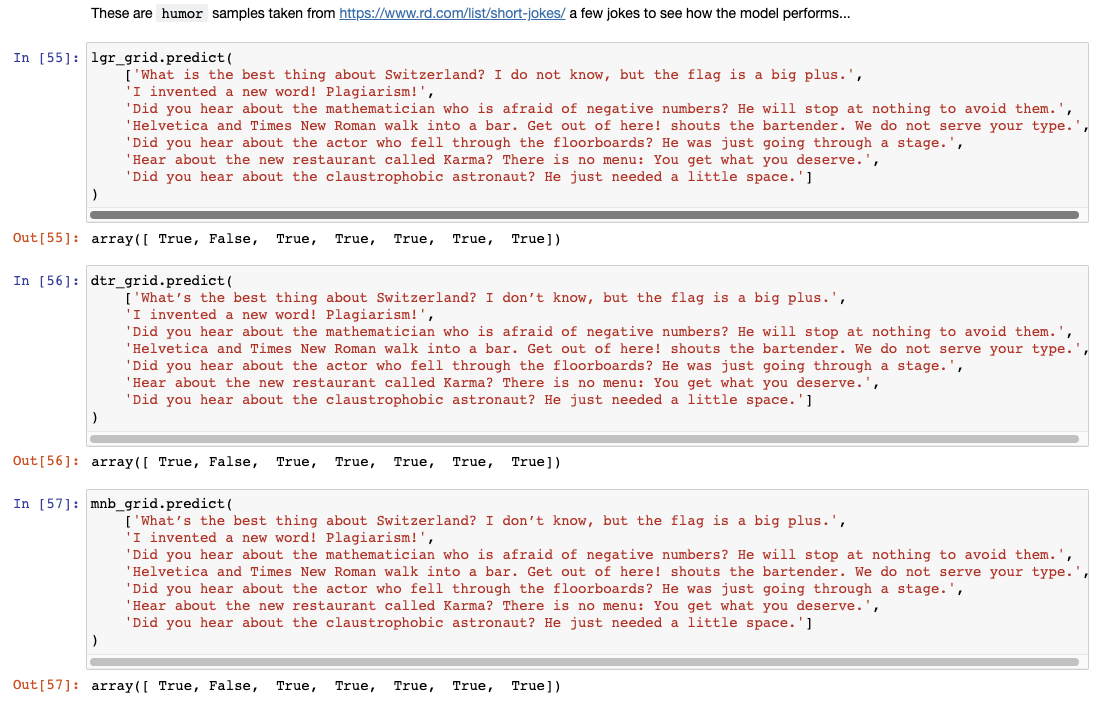


Ordinary Sentences:

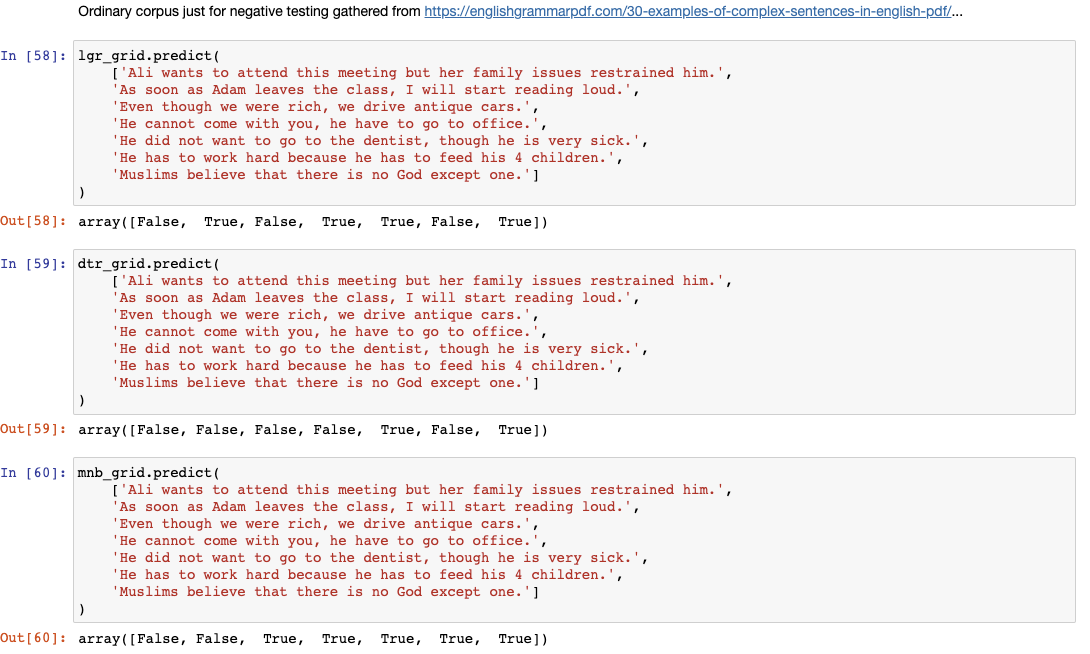


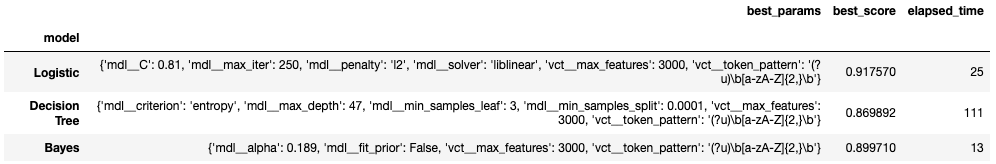
Old result output:

Humor:



Ordinary Sentences:





**Results w/750 features**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **model** | **vectorizer** | **best\_params** | **best\_score** | **time** | **pipe** | **grid** | **train\_acc** | **test\_acc** | **cv\_results** |
| **0** | Logistic | CountVectorizer() | {'vct\_\_max\_features': 3000} | 0.917900 | 5.992758 | (CountVectorizer(), LogisticRegression(max\_iter=250)) | GridSearchCV(estimator=Pipeline(steps=[('vct', CountVectorizer()),\n ('mdl',\n LogisticRegression(max\_iter=250))]),\n param\_grid={'vct\_\_max\_features': [2000, 3000]}) | 0.926327 | 0.91828 | {'mean\_fit\_time': [2.7495930194854736, 2.7231404304504396], 'std\_fit\_time': [0.10075718557172608, 0.04788366726253731], 'mean\_score\_time': [0.25781950950622556, 0.2622048377990723], 'std\_score\_time': [0.016999828906807662, 0.017495637249437063], 'param\_vct\_\_max\_features': [2000, 3000], 'params': [{'vct\_\_max\_features': 2000}, {'vct\_\_max\_features': 3000}], 'split0\_test\_score': [0.9146333333333333, 0.9184666666666667], 'split1\_test\_score': [0.9144, 0.9180333333333334], 'split2\_test\_score': [0.9129333333333334, 0.9184], 'split3\_test\_score': [0.9134, 0.9180333333333334], 'split4\_test\_score': [0.9125333333333333, 0.9165666666666666], 'mean\_test\_score': [0.91358, 0.9179], 'std\_test\_score': [0.0008158431221748353, 0.0006905714220041996], 'rank\_test\_score': [2, 1]} |
| **1** | Decision Tree | CountVectorizer() | {'vct\_\_max\_features': 3000} | 0.868007 | 49.914065 | (CountVectorizer(), DecisionTreeClassifier()) | GridSearchCV(estimator=Pipeline(steps=[('vct', CountVectorizer()),\n ('mdl', DecisionTreeClassifier())]),\n param\_grid={'vct\_\_max\_features': [2000, 3000]}) | 0.999980 | 0.86980 | {'mean\_fit\_time': [23.27494058609009, 26.054628562927245], 'std\_fit\_time': [1.1582656254838002, 1.002077955691884], 'mean\_score\_time': [0.29335222244262693, 0.2911435604095459], 'std\_score\_time': [0.028331250131009397, 0.027771584810528174], 'param\_vct\_\_max\_features': [2000, 3000], 'params': [{'vct\_\_max\_features': 2000}, {'vct\_\_max\_features': 3000}], 'split0\_test\_score': [0.8667333333333334, 0.8664333333333334], 'split1\_test\_score': [0.8661333333333333, 0.8686333333333334], 'split2\_test\_score': [0.8675666666666667, 0.8690666666666667], 'split3\_test\_score': [0.8695333333333334, 0.8700666666666667], 'split4\_test\_score': [0.8648333333333333, 0.8658333333333333], 'mean\_test\_score': [0.8669600000000001, 0.8680066666666667], 'std\_test\_score': [0.0015650914066313083, 0.001609886124337148], 'rank\_test\_score': [2, 1]} |
| **2** | Bayes | CountVectorizer() | {'vct\_\_max\_features': 3000} | 0.897707 | 2.860348 | (CountVectorizer(), MultinomialNB()) | GridSearchCV(estimator=Pipeline(steps=[('vct', CountVectorizer()),\n ('mdl', MultinomialNB())]),\n param\_grid={'vct\_\_max\_features': [2000, 3000]}) | 0.899933 | 0.89748 | {'mean\_fit\_time': [1.1819184303283692, 1.141023874282837], 'std\_fit\_time': [0.051990040089916395, 0.052240310197834594], 'mean\_score\_time': [0.26781587600708007, 0.26958932876586916], 'std\_score\_time': [0.014822820900283308, 0.015183185567084258], 'param\_vct\_\_max\_features': [2000, 3000], 'params': [{'vct\_\_max\_features': 2000}, {'vct\_\_max\_features': 3000}], 'split0\_test\_score': [0.8929333333333334, 0.8989], 'split1\_test\_score': [0.8918666666666667, 0.8987], 'split2\_test\_score': [0.8919, 0.8979333333333334], 'split3\_test\_score': [0.8899666666666667, 0.8980666666666667], 'split4\_test\_score': [0.8884666666666666, 0.8949333333333334], 'mean\_test\_score': [0.8910266666666666, 0.8977066666666668], 'std\_test\_score': [0.0015988051093794543, 0.0014340928220384606], 'rank\_test\_score': [2, 1]} |
| **3** | Logistic | TfidfVectorizer() | {'vct\_\_max\_features': 3000} | 0.914020 | 4.403639 | (TfidfVectorizer(), LogisticRegression(max\_iter=250)) | GridSearchCV(estimator=Pipeline(steps=[('vct', TfidfVectorizer()),\n ('mdl',\n LogisticRegression(max\_iter=250))]),\n param\_grid={'vct\_\_max\_features': [2000, 3000]}) | 0.920947 | 0.91390 | {'mean\_fit\_time': [1.792166566848755, 2.089572286605835], 'std\_fit\_time': [0.09471320884755943, 0.1354018449832842], 'mean\_score\_time': [0.2589530944824219, 0.26294713020324706], 'std\_score\_time': [0.008737283702478605, 0.0065056741860597825], 'param\_vct\_\_max\_features': [2000, 3000], 'params': [{'vct\_\_max\_features': 2000}, {'vct\_\_max\_features': 3000}], 'split0\_test\_score': [0.9112666666666667, 0.9149666666666667], 'split1\_test\_score': [0.9105666666666666, 0.9143], 'split2\_test\_score': [0.9102333333333333, 0.9149666666666667], 'split3\_test\_score': [0.9101, 0.9149666666666667], 'split4\_test\_score': [0.9075333333333333, 0.9109], 'mean\_test\_score': [0.90994, 0.91402], 'std\_test\_score': [0.0012693655458097615, 0.0015812231552398471], 'rank\_test\_score': [2, 1]} |
| **4** | Decision Tree | TfidfVectorizer() | {'vct\_\_max\_features': 3000} | 0.858920 | 64.755211 | (TfidfVectorizer(), DecisionTreeClassifier()) | GridSearchCV(estimator=Pipeline(steps=[('vct', TfidfVectorizer()),\n ('mdl', DecisionTreeClassifier())]),\n param\_grid={'vct\_\_max\_features': [2000, 3000]}) | 0.999973 | 0.86040 | {'mean\_fit\_time': [30.553899383544923, 33.60998067855835], 'std\_fit\_time': [1.2418354035192003, 1.1120848463730222], 'mean\_score\_time': [0.28730478286743166, 0.3040257453918457], 'std\_score\_time': [0.009886671305487493, 0.0322357586832567], 'param\_vct\_\_max\_features': [2000, 3000], 'params': [{'vct\_\_max\_features': 2000}, {'vct\_\_max\_features': 3000}], 'split0\_test\_score': [0.8556666666666667, 0.859], 'split1\_test\_score': [0.8593666666666666, 0.8598333333333333], 'split2\_test\_score': [0.8591, 0.8588], 'split3\_test\_score': [0.8578666666666667, 0.8612333333333333], 'split4\_test\_score': [0.8553666666666667, 0.8557333333333333], 'mean\_test\_score': [0.8574733333333333, 0.85892], 'std\_test\_score': [0.001678544342908758, 0.0018089407581970774], 'rank\_test\_score': [2, 1]} |
| **5** | Bayes | TfidfVectorizer() | {'vct\_\_max\_features': 3000} | 0.893347 | 3.314120 | (TfidfVectorizer(), MultinomialNB()) | GridSearchCV(estimator=Pipeline(steps=[('vct', TfidfVectorizer()),\n ('mdl', MultinomialNB())]),\n param\_grid={'vct\_\_max\_features': [2000, 3000]}) | 0.896233 | 0.89312 | {'mean\_fit\_time': [1.3477141857147217, 1.3426270484924316], 'std\_fit\_time': [0.012959185003089737, 0.006836603724063561], 'mean\_score\_time': [0.3094798564910889, 0.3142988204956055], 'std\_score\_time': [0.0022928169241420693, 0.006474015426677336], 'param\_vct\_\_max\_features': [2000, 3000], 'params': [{'vct\_\_max\_features': 2000}, {'vct\_\_max\_features': 3000}], 'split0\_test\_score': [0.8868333333333334, 0.8924666666666666], 'split1\_test\_score': [0.8875333333333333, 0.8948666666666667], 'split2\_test\_score': [0.8870666666666667, 0.8951], 'split3\_test\_score': [0.8851333333333333, 0.8935], 'split4\_test\_score': [0.8833666666666666, 0.8908], 'mean\_test\_score': [0.8859866666666665, 0.8933466666666666], 'std\_test\_score': [0.0015406492138056666, 0.0015917286200857225], 'rank\_test\_score': [2, 1]} |