**Berkeley 18 - 21 AI/ML Certificate Program Modules 18 - 21**



Prepare Journey Data, generalize and mask entries.!

Slack: Here's the announcement pointing to the Slack space.

Topic: Student Success Coaching and Slack Workspace Now Available!

(emeritus.org)

<https://student.emeritus.org/courses/4765/discussion_topics/257315>

**Savio’s Colab:**

<https://colab.research.google.com/drive/1m640WoWTbTgw0ymZoIw9K0wyCfMXtbYH#scrollTo=YdnBR4JQ_MMA>

One 1:1 Meeting per participant across the period of Weeks 12-15

June 8 - July 5 - Done!

One 1:1 Meeting per participant across the period of Modules 21-23

**August 17 - September 6**

<https://student.emeritus.org/courses/4765/pages/how-to-schedule-a-1-1-session?module_item_id=999645>

<https://student.emeritus.org/courses/4765/pages/capstone-project-overview?module_item_id=999644>

<https://student.emeritus.org>

**An outline of the BH-PCMLAI program calendar**

|  |  |  |  |
| --- | --- | --- | --- |
| **Module #** | **Module Title** | **Week #** | **Date** |
| **0** | Program Orientation | 0 | **Wednesday, March 02, 2022** |
| **1** | Introduction to Machine Learning | 1 | **Wednesday, March 09, 2022** |
| **2** | Fundamentals of Machine Learning | 2 | **Wednesday, March 16, 2022** |
| **3** | Introduction to Data Analysis | 3 | **Wednesday, March 23, 2022** |
| **4** | Fundamentals of Data Analysis | 4 | **Wednesday, March 30, 2022** |
| **5** | Practical Application 1 | 5 | **Wednesday, April 06, 2022** |
| Break Week |  |  | **Wednesday, April 13, 2022** |
| **6** | Clustering and Principal Component Analysis (PCA) | 6 | **Wednesday, April 20, 2022** |
| **7** | Linear and Multiple Regressions | 7 | **Wednesday, April 27, 2022** |
| **8** | Feature Engineering and Overfitting | 8 | **Wednesday, May 04, 2022** |
| **9** | Model Selection and Regularization | 9 | **Wednesday, May 11, 2022** |
| **10** | Time Series Analysis and Forecasting | 10 | **Wednesday, May 18, 2022** |
| **11** | Practical Application 2 | 11 | **Wednesday, May 25, 2022** |
| Break Week |  |  | **Wednesday, June 01, 2022** |
| **12** | Classification and k-Nearest Neighbors (KNN) | 12 | **Wednesday, June 08, 2022** |
| **13** | Logistic Regression | 13 | **Wednesday, June 15, 2022** |
| **14** | Decision Trees | 14 | **Wednesday, June 22, 2022** |
| **15** | Gradient Descent and Optimization | 15 | **Wednesday, June 29, 2022** |
| **16** | Support Vector Machines (SVMs) | 16 | **Wednesday, July 06, 2022** |
| **17** | Practical Application 3 | 17 | **Wednesday, July 13, 2022** |
| Break Week |  |  | **Wednesday, July 20, 2022** |
| **18** | Natural Language Procession (NLP) | 18 | **Wednesday, July 27, 2022** |
| **19** | Recommendation Systems | 19 | **Wednesday, August 03, 2022** |
| **20** | Capstone 1 | 20 | **Wednesday, August 10, 2022** |
| **21** | Ensemble Techniques (GBM, XGB, and Random Forest) | 21 | **Wednesday, August 17, 2022** |
| **22** | Deep Neural Networks 1 | 22 | **Wednesday, August 24, 2022** |
| **23** | Deep Neural Networks 2 | 23 | **Wednesday, August 31, 2022** |
| **24** | Capstone 2 | 24 | **Wednesday, September 07, 2022** |

 ————— o —————

 ————— o —————

**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:

1. **NLU** refers to analyzing a sentence's meaning based on the syntactic and semantic elements of the sentence
2. **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:

* Tokenization
* Lower casing
* Stop words removal
* Stemming
* 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:

1. **Intrinsic evaluation** focuses on intermediate objectives (e.g., the performance of an NLP component on a specific subtask)
2. **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*.*

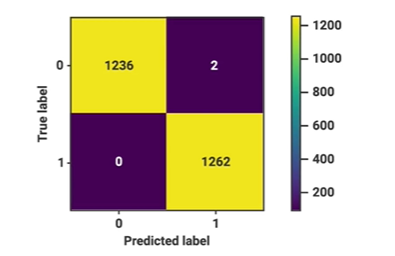
Which of the following techniques are part of preprocessing text data? *Check all that apply. :* StopWords, Stemmer, Tokenizer

*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**

* **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.
* **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,

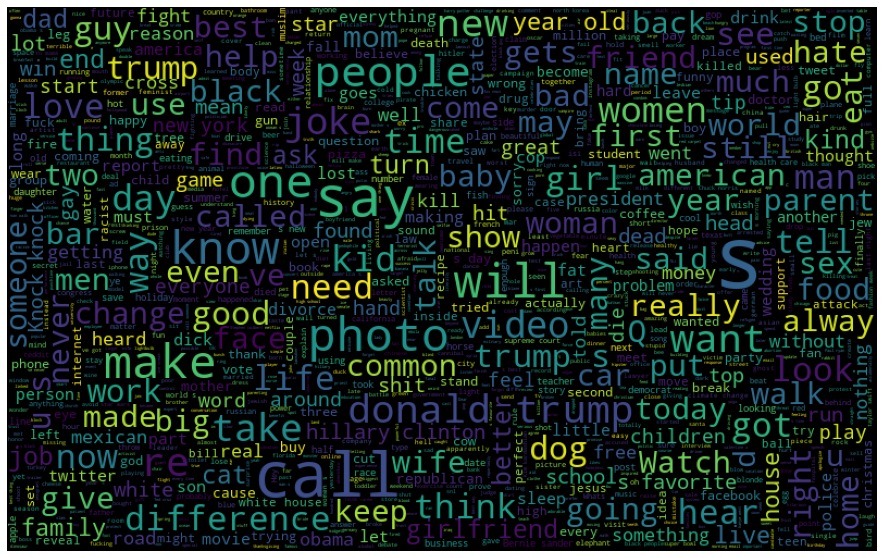
width = 800, height = 500,

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')

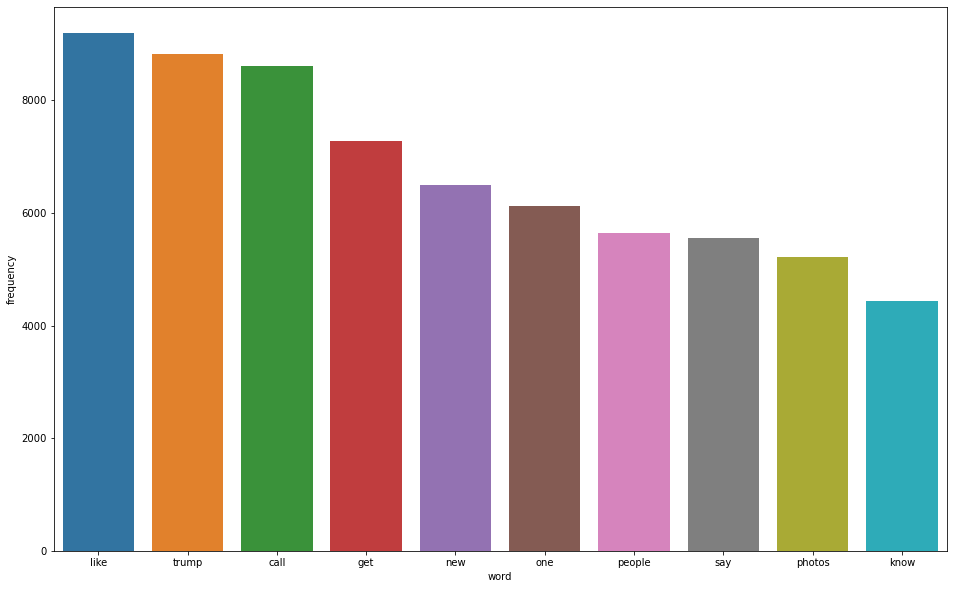
freq = freq[~freq['word'].str.lower().isin(stop)]

# 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)')

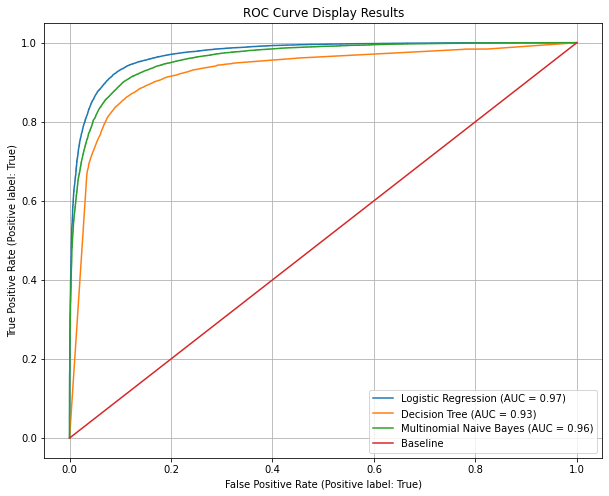
RocCurveDisplay.from\_estimator(mnb\_grid, X\_test, y\_test, pos\_label = 1, ax = ax, label = 'Multinomial Naive Bayes (AUC = 0.96)')

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)

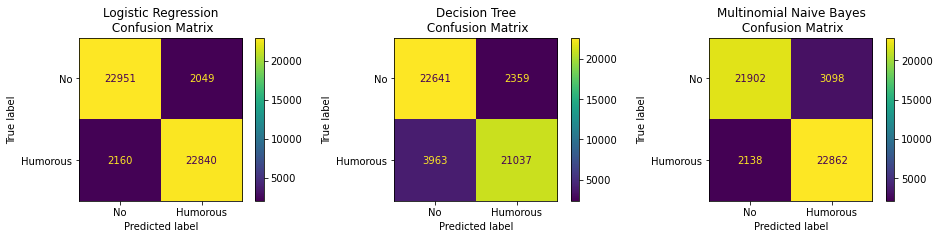
ax[2].grid(False)

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')

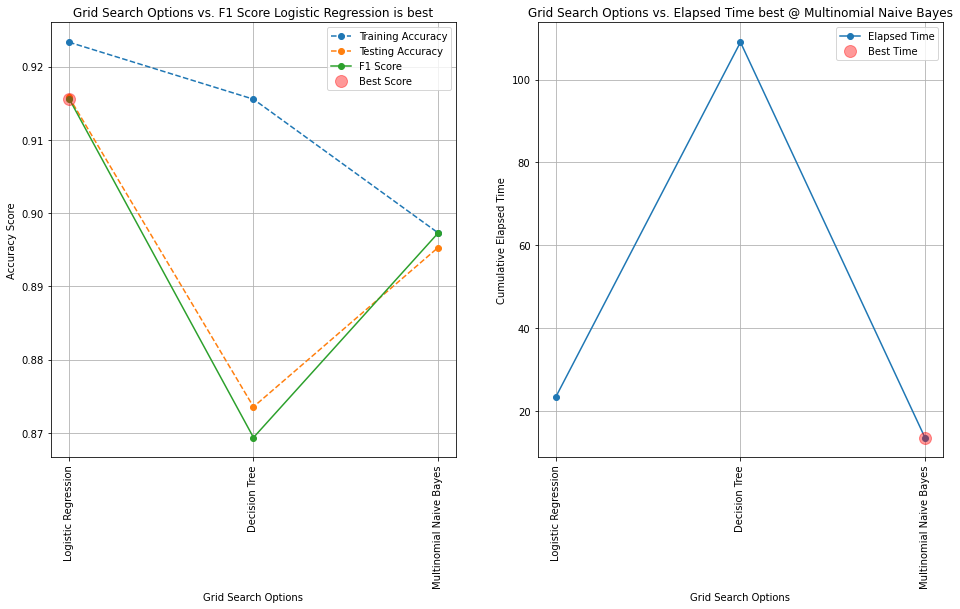
ax[1].set\_ylabel('Cumulative Elapsed Time')

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

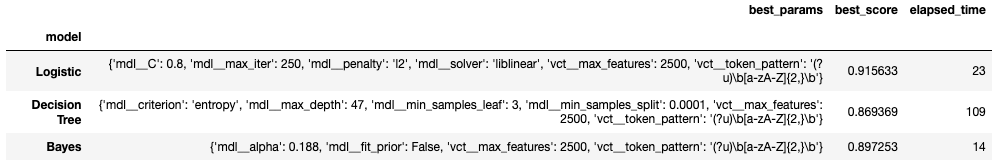
ax[1].plot(np.argmin(elapsed\_times), min(elapsed\_times), 'ro', markersize = 12, alpha = 0.4, label = 'Best Time')

ax[1].legend()

ax[1].grid(True)



**Result Summary Table**



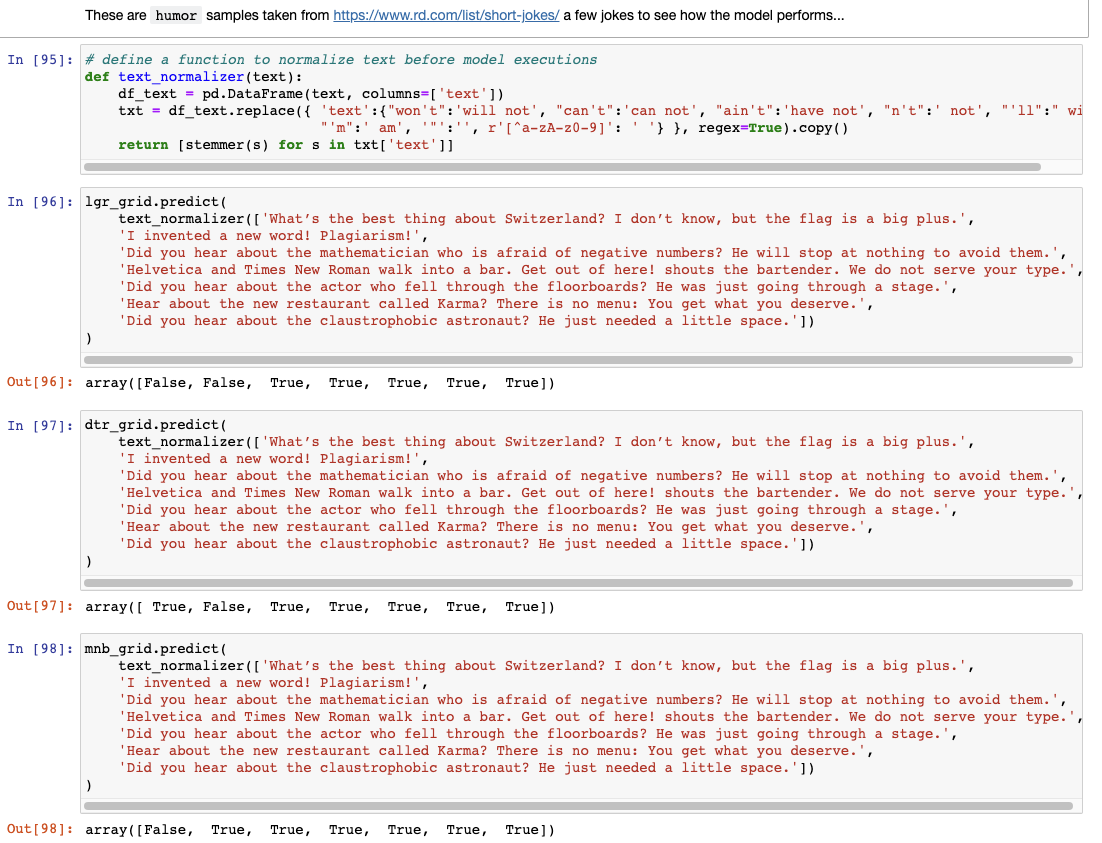
**Conclusion**

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:

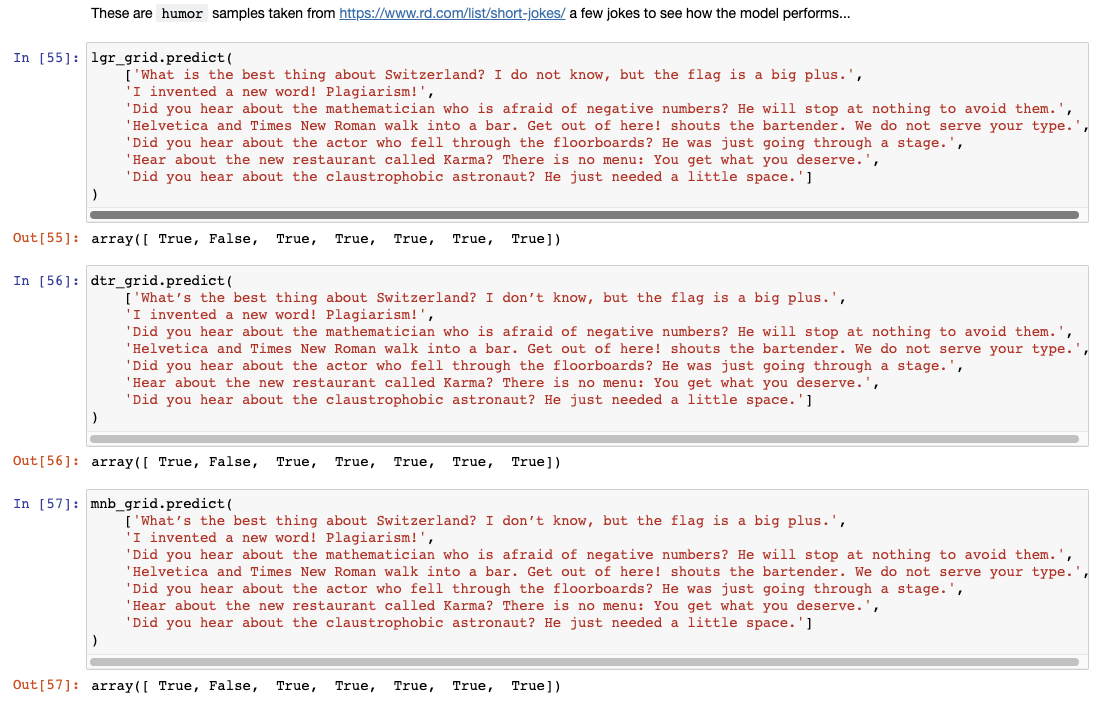


Ordinary Sentences:

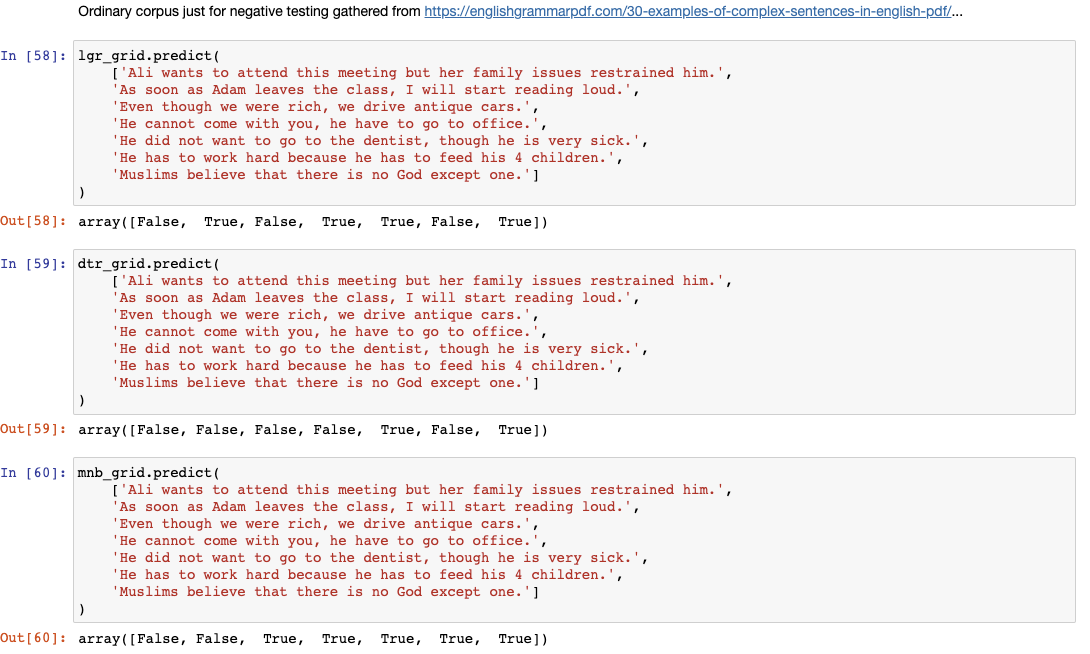


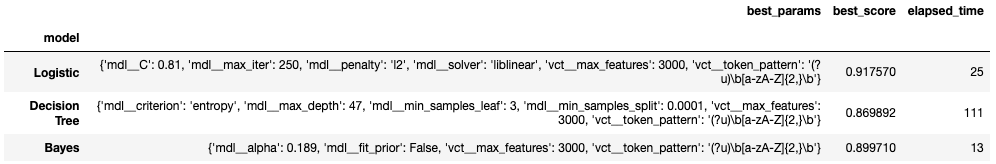
Old result output:

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]} |

 ————— o —————

**Module 19**

**Recommendation Systems**

Many large companies, such as Google, Instagram, Spotify, Amazon, Netflix, etc., use recommendation systems to increase user engagement. Spotify, for example, recommends songs similar to those you have liked or listened to so that you will continue listening to music on their platform. Likewise, Amazon recommends products to its users based on the data they have collected on them.

* [Video Transcripts](https://student.emeritus.org/courses/4765/files/3548093?wrap=1)
* [Download Video Transcripts](https://student.emeritus.org/courses/4765/files/3548093/download?download_frd=1)
* [Quick Reference Guide](https://student.emeritus.org/courses/4765/files/3548094?wrap=1)
* [Download Quick Reference Guide](https://student.emeritus.org/courses/4765/files/3548094/download?download_frd=1)

**Glossary**

**Collaborative Filtering**

A method of making automatic predictions about the interests of a user by collecting preferences or taste information from many users

**Content-Based Filtering**

A method of making recommendations based on user preferences for product features

**Funk SVD**

A stochastic gradient descent process developed by Simon Funk that minimizes the error on the known values

**Matrix Factorization**

A type of algorithm that works by decomposing the user-item interaction matrix into the product of two lower-dimensionality rectangular matrices

**SVD**

The abbreviation for singular value decomposition

**Install Surprise:**

Make sure “admin” on Mac!

pip install scikit-surprise

pip install -U --trusted-host pypi.org --trusted-host files.pythonhosted.org scikit-surprise

Manish:

**conda install -c conda-forge scikit-surprise**

**Matilde’s Session:**

**Savio’s Session:**

<https://github.com/kelvins/awesome-mlops>

<https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.linear_kernel.html>

He works for Densu? Marketing purpose: cosine similarity they use

**Notes:**

**Recommendation Systems**

A recommendation engine is a machine learning algorithm that ranks or rates products based on certain criteria. As a general definition, a recommender system is a system that predicts the rating a user will give an item. The predictions are then ranked and returned to the user.

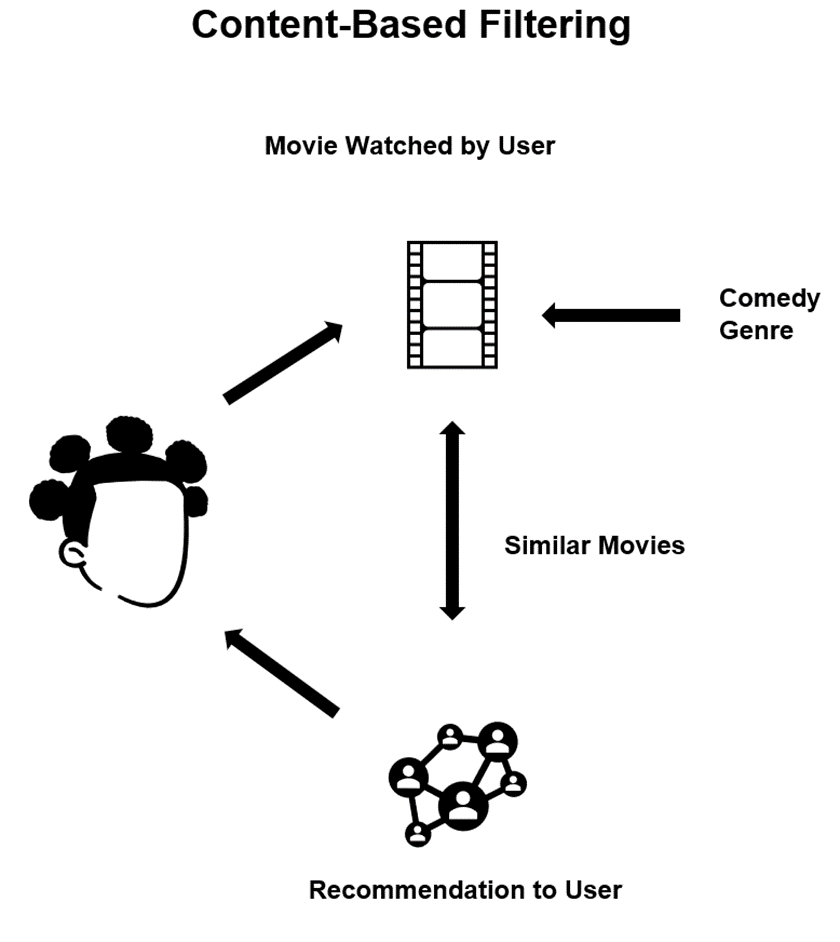
Recommender systems fall into three main categories:

1. **Content-based filtering** uses similarities in the features of products, services, and content to make recommendations
2. **Collaborative filtering**uses similar users' preferences to provide recommendations to a particular user
3. **Hybrid recommender systems** combine multiple recommender strategies to make recommendations, using the advantages of each in different ways

**Content-Based Filtering**

Content-based filtering is a technique that takes advantage of similarities in features to make decisions. Typically, this technique is applied to recommender systems, which are algorithms used to suggest things to users based on their experience.

In this method, a user's interests are compared to the product's features. Those products with the most overlap between their features and user interests will be recommended, as shown in Figure 1 below.



r̂ is equal to row (P) ⋅ col (Q). i,𝑗 ij

2 We can give the squared error of a prediction 𝑒𝑖,𝑗, as the difference between

2

(𝑟̂ − 𝑟 )

**Matrix Factorization**

The matrix factorization method generates latent features when multiplied by two entities of different types. Matrix factorization identifies the relationship between items' and users' entities in collaborative filtering. Using the input of users' ratings on the shop items, you hope to predict how the users will rate the items so the users can receive recommendations based on the prediction.

<https://www.youtube.com/watch?v=ZspR5PZemcs>

**Other Matrix Factorization Techniques**

In this module, you have learned some matrix factorization techniques for recommender systems. These techniques discover latent features in users and items. With this method, cold start problems and data sparsity can be reduced. This lesson will introduce you to some other techniques for matrix factorization methods, such as:

* Singular value decomposition (SVD)
* Probabilistic matrix factorization (PMF)
* Non-negative matrix factorization (NMF)
* Bayesian probabilistic matrix factorization (BPMF)

**Singular Value Decomposition (SVD)**

The famous SVD algorithm, as popularized by Simon Funk during the Netflix Prize, is a technique for reducing dimensions and creating superior quality recommendations for users. The SVD is commonly used to produce low-rank approximations before computing neighborhoods in collaborative filtering. SVD can also be used in collaborative filtering to discover latent associations between users and items to predict the probability of users selecting certain things.

SVD has also been successful with the following variants:

* FunkSVD
  1. Simon Funk's original algorithm factored the user-item rating matrix as the product of two lower dimensional matrices
* SVD++
  1. The SVD++ algorithm is an optimized SVD algorithm designed to provide implicit feedback to improve prediction accuracy
* Regularized SVD
  1. RSVD is a fast probability-based algorithm that can compute the near-optimal low-rank singular value decomposition of vast amounts of data with high accuracy. The idea of RSVD is to represent the data as compressed to capture the essential information. From this compressed representation, a low-rank singular value decomposition can be derived.
* Iterative SVD
  1. ISVD is used to estimate the singular value decomposition of an incomplete given matrix. ISVD uses first-order optimization over orthogonal manifolds and automatically estimates the rank of SVD. The purpose here is to estimate the singular vectors by optimizing the suitable space, which is the space of the orthogonal matrix manifolds.

**Probabilistic Matrix Factorization (PMF)**

The PMF approach assumes that a small number of unobserved factors determine a user's attitudes and preferences. User preferences are modeled by linearly combining item factor vectors with user-specific coefficients in a linear factor model. This technique has proven successful on huge, sparse, and imbalanced datasets.

**Non-negative Matrix Factorization (NMF)**

In NMF, the principal components of a set of non-negative data vectors are automatically extracted. The principal components can be used to extract significant and sparse features from those vectors.

NMF can reduce prediction errors compared to other techniques, such as SVD. Furthermore, when used in collaborative filtering, the NMF technique always leads to interpretable and sparse decompositions of non-negative matrices.

**Bayesian Probabilistic Matrix Factorization (BPMF)**

BPMF is a model in which capacity is automatically controlled by integrating all model parameters and hyperparameters, thereby allowing it to avoid parameter tuning and providing predictive distribution. Thus, the concept of BPMF is extended to recommendations where top N queries are recommended to users. This allows for more efficient and accurate predictions.

**The SURPRISE Library**

Python's SURPRISE module allows the creation and testing of rate prediction algorithms. Users familiar with the Python machine learning ecosystem should feel at home using this library since it closely resembles the scikit-learn API. SURPRISE provides a set of estimators (or prediction algorithms) for evaluating predictions. In addition, there are implementations of classical algorithms like SVD and NMF and similarity-based algorithms.

Furthermore, the SURPRISE library includes tools for model evaluation, including cross-validation iterators and learned metrics built into scikit-learn, as well as automatic hyperparameter search and grid search for model selection. Finally, a light API allows users to develop their recommendation technique with fewer lines of code.

**Hybrid Recommender Systems**

As you have learned, recommender systems are software tools that generate and present the user with suggested items and other entities based on various strategies. A hybrid recommender system combines multiple recommendation strategies to take advantage of their complementary attributes.

As an example, by combining collaborative and content-based filtering, you may overcome some of the shortcomings faced when each method is used separately. You can implement hybrid recommender system approaches by using content-based and collaborative methods to generate predictions individually, then combining them, or you can simply add the capabilities of collaborative methods to a content-based approach, as shown in Figure 3 below.

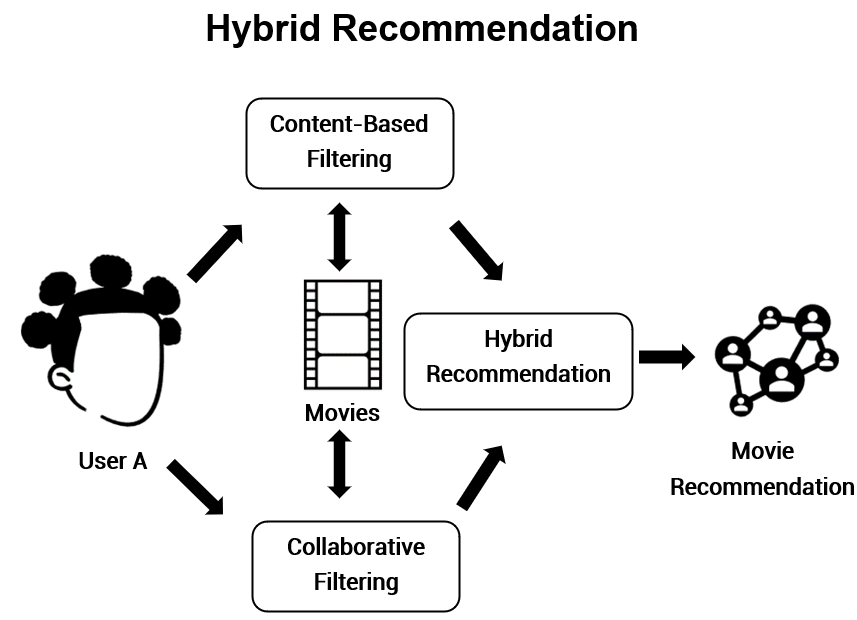


Figure 3

**Module Issues:**

**Codio 19.1 Problem 1**: The description is not clear that prediction must be done for **Dropdead** missing value.

**Codio 19.1 Problem 5**:Truncate output to 2 decimal places for just this entry to pass the hidden test: reviews\_df\_full.loc[['Dropdead'], 'Mandy'] = np.floor(reviews\_df\_full.loc[['Dropdead'], 'Mandy'][0]\*100)/100.0

**Codio 19.2**: It did not grade first so I had to upload the grading script!

**Codio 19.4**: It did not grade first so I had to upload the grading script!

**Codio 19.6 Problem 1**: It does not specify what columns are needed as ’userId', 'title', 'rating’.

**Codio 19.7 Problem 7**: The hidden test generates different result although everything is the same, to by-pass assign these variables:

slope\_one\_preds\_ = slope\_one\_preds

svd\_preds\_ = svd\_preds

**Quizes:**

A recommender system seeks to predict the "rating" or "preference" a user would give an item. : True

*You are correct! The answer “*True*” is correct because recommender systems predict the "rating" or "preference" a user would give an item.*

When recommending new items that users have never engaged with, the platform does not know the ratings in advance. In this instance, what information will the platform use to make recommendations for this item? : Everything it knows about users

*You are correct! The answer “*Everything it knows about users*” is correct because when the platform does not know the ratings in advance, it will attempt to recommend the item based on everything it knows about its users.*

In a recommender system, the item factors are factors that describe the first item. : False

*You are correct! The answer “*False*” is correct because, in a recommender system, the item factors are factors that describe each item.*

Imagine that you are using the item factors “lo-fi\_indie” and “slick\_pop” and the user rating to build a linear regression model. The parameters that you get from linear regression are θ₁ and θ₂. In the context of the recommender system, what are these parameters known as? : User factors

*You are correct! The answer “*User factors*” is correct because the parameters θ₁ and θ₂ measured from linear regression are called user factors.*

In recommender systems terminology, the predictions made by applying linear regression to the item factors and ratings data is given as the dot product of user factors with the item’s factors. : False

*You are correct! The answer “*False*” is correct because the prediction is the dot product of user factors with the item’s factors plus the bias term.*

One big problem in content-based filtering is that in real-world scenarios, the items are not typically categorized into content categories. : True

*You are correct! The answer “*True*” is correct because the problem in content-based filtering is that in the real world, not all items are categorized into content categories.*

In terms of recommender systems, how can the user factors be calculated? : Parameters of linear regression

*You are correct! The answer “*Parameters of linear regression*” is correct because the parameters of the linear regression model that were built using item factors are the user factors.*

To guess the unknown rating from users using collaborative filtering, the algorithm starts with declaring random user factors. : False

*You are correct! The answer “*False*” is correct because the collaborative filtering algorithm starts with declaring random item factors.*

To calculate the user factors for all users from randomly-defined item factors, which model is used? : Linear regression

*You are correct! The answer “*Linear regression*” is correct because the linear regression parameters are the user factors for the collaborative filtering approach.*

To recompute the item factors from the inferred user factors in the collaborative filtering approach, you build a linear regression model using the inferred user factors for each item to get the new inferred item factors. : True

*You are correct! The answer “*True*” is correct because you would build a linear regression model using the inferred user factors for each item to get the new inferred item factors.*

A drawback to the collaborative filtering approach is that it must run through all the item and user factors. : True

*You are correct! The answer “*True*” is correct because the collaborative filtering approach must run through all the item and user factors.*

What is the formula for the predicted rating r^i,j using the item factor matrix and the user factors matrix? : r^i,j= row*i* (P)⋅col*j*(Q)

*You are correct! The answer “*r^i,j= row*i* (P)⋅col*j*(Q)*” is correct because this is the correct formula for predicted ratings.*

The mean squared error formula for collaborative filtering approach is:

MSE=∑i=1M∑jϵRiN(rowi(P)⋅colj(Q)−ri,j)2

The symbol “*Ri*” represents all the items which user *i* has rated. : True

*You are correct! The answer “*True*” is correct because the symbol “Ri” represents all the items which user i has rated.*

Consider the user factors and item factors provided in the tables below.

*Q: Z × N*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Ommazh** | **Melt-Banana** | **BTS** | **Zhou Shen** | **Sanam** |
| F1 | **−**1.53 | **−**1.77 | 1.16 | 1.09 | 0.73 |
| F2 | 1.77 | 1.88 | 1.45 | 1.74 | 1.69 |

*P: M × Z*

|  |  |  |
| --- | --- | --- |
|  | **F1** | **F2** |
| An | **−**1.52 | 1.35 |
| Bhavana | **−**1.51 | 1.11 |
| Cordelia | 1.15 | 2.16 |
| Diego | 1.19 | 2.15 |

Given the user factors and item factors, what would be user “An’s” rating for item “Ommazh”? : −1.53 × −1.52 + 1.77 × 1.35

*You are correct! The answer “*−1.53 × −1.52 + 1.77 × 1.35*” is correct because the user “An’s” rating for item “Ommazh” would be the dot product of Ommazh column and An’s row.*

If the number of users in the data are four and the items in the data are five, considering 50 factors to be formed, what would be the dimensions of user factors matrix *P* and item factors matrix *Q*? : P: 4 × 50

Q: 50 × 5

*You are correct! The answer “*P: 4 × 50

Q: 50 × 5*” is correct because the dimensions for the user factor matrix are “P: M × Z” and the dimensions for the item factors matrix are “Q: Z × N”.*

The algorithm Funk SVD decomposes an M × N matrix into three matrices of sizes M × M, M × N, and N × N. : False

*You are correct! The answer “*False*” is correct because the algorithm Funk SVD decomposes into two matrices of size M × Z and Z × N, where Z is any size of users' choosing.*

Real SVD cannot be used for matrix decomposition because SVD does not have any way to deal with missing entries. : True

*You are correct! The answer “*True*” is correct because the SVD doesn't work with missing values.*

Scikit-learn supports the Funk SVD matrix decomposition algorithm. : False

*You are correct! The answer “*False*” is correct because scikit-learn does not support the Funk SVD matrix decomposition algorithm.*

The constructor used in the surprise.SVD() function to declare the number of factors is “n\_epochs”. : False

*You are correct! The answer “*False*” is correct because the constructor used in the*surprise.SVD()*function to declare the number of factors is “n\_factors”.*

Considering the SVD algorithm in the Python library, is the given statement correct? : Incorrect

*You are correct! The answer “*Incorrect*” is correct because the function*model.test()*gives results only for test sets. It does not work on objects of type training set.*

In an SVD model built using the SURPRISE library, the statement model.pu is used to get the user factors and the statement model.qi is used to get the item factors. : True

*You are correct! The answer “*True*” is correct because the statement*model.pu*is used to get the user factors and the statement*model.qi*is used to get the item factors.*

What does the given Python statement provide?

model.pu @ model.qi.T : Predictions

*You are correct! The answer “*Predictions*” is correct because the given statement provides the predictions of the model by dot product of user factors with the transpose of the item factors.*

Which of the following can be considered metrics for recommender systems? (*Check all that apply.) :* Promoted by advertisers, Items that are popular on the service, Predicted to have high ratings

*You are correct! The answers “*Predicted to have high ratings,*” “*Promoted by advertisers,*” and “*Items that are popular on the service*” are correct because these are considered the metrics for recommender systems.*

What are the systems that use a combination of metrics for recommendation called? : Hybrid recommender systems

*You are correct! The answer “*Hybrid recommender systems*” is correct because the systems that use a combination of metrics for recommendation are called hybrid recommender systems.*

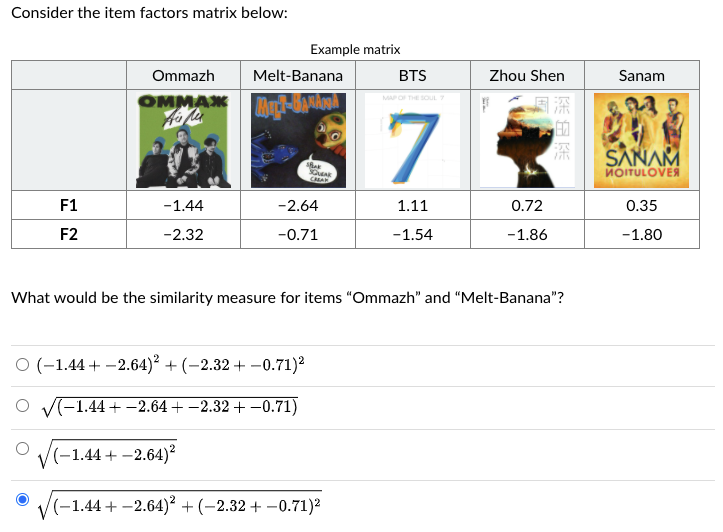
Consider an item factors matrix that tells you there are two factor values for five different items. What does Z equal in this instance? : 2

*You are correct! The answer “*2*” is correct because the value Z that represents the dimensions is equal to the number of factors.*

Consider the item factors matrix below:

Example matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Ommazh | Melt-Banana | BTS | Zhou Shen | Sanam |
|  |  |  |  |  |  |
| **F1** | −1.44 | −2.64 | 1.11 | 0.72 | 0.35 |
| **F2** | −2.32 | −0.71 | −1.54 | −1.86 | −1.80 |



What would be the similarity measure for items “Ommazh” and “Melt-Banana”?

*You are correct! The answer “sqrt(*(−1.44+−2.64)^2+(−2.32+−0.71)^2)*” is correct because the similarity is simply the distance between the two points in Z dimensional space where Z is equal to the number of factors.*

**Try-It Activity 19.1: Building a Recommender System with SURPRISE - Section B**

**Introduction to Dataset**

**ratings.csv** in ml-latest-small file set taken from [grouplens](https://grouplens.org/datasets/movielens/) describes 5-star rating and free-text tagging activity from MovieLens. It contains **100836** ratings across 9742 movies were created by 610 users between March 29, 1996 and September 24, 2018 at random for inclusion who had rated at least 20 movies.

**Ratings Data File Structure (ratings.csv)**

Each line of this file represents one rating of one movie by one user, and has the following format:

userId,movieId,rating,timestamp

The lines within this file are ordered first by userId, then, within user, by movieId.

Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars).

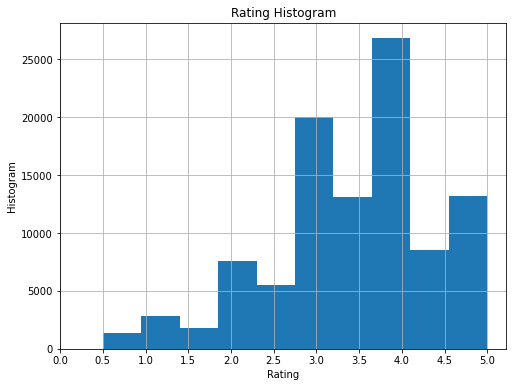
Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

User Ids have been anonymized but they are consistent across files.

Movie Ids are consistent across files and with those used on the MovieLens web site (e.g., id 1 corresponds to the URL <https://movielens.org/movies/1>). Only movies with at least one rating or tag are included in the dataset.

**Exploratory Data Analysis**

No preprocessing is needed, *rating* in the range of 0 and 5, the histogram as follows:



Dataset were prepared for model executions:

# set rating scale to 0 - 5

reader = Reader(rating\_scale=(0, 5))

# reformat data

data = Dataset.load\_from\_df(df[['movieId', 'userId', 'rating']], reader)

# build train and test sets

train = data.build\_full\_trainset()

test = train.build\_testset()

KNNBasic, SVD, NMF, SlopeOne, and CoClustering were built, fit, predicted and cross-validated:

%%time

knnb = KNNBasic(random\_state = 93)

knnb.fit(train)

# predictions

knnb\_preds = knnb.test(test)

# cross validations

knnb\_cross = cross\_validate(knnb, data, measures=['RMSE'], verbose=True)

%%time

fsvd = SVD(random\_state = 93)

fsvd.fit(train)

# predictions

fsvd\_preds = fsvd.test(test)

# cross validations

fsvd\_cross = cross\_validate(fsvd, data, measures=['RMSE'], verbose=True)

%%time

nmf = NMF(random\_state = 93)

nmf.fit(train)

# predictions

nmf\_preds = nmf.test(test)

# cross validations

nmf\_cross = cross\_validate(nmf, data, measures=['RMSE'], verbose=True)

%%time

sone = SlopeOne()

sone.fit(train)

# predictions

sone\_preds = sone.test(test)

# cross validations

sone\_cross = cross\_validate(sone, data, measures=['RMSE'], verbose=True)

%%time

cocl = CoClustering(random\_state = 93)

cocl.fit(train)

# predictions

cocl\_preds = cocl.test(test)

# cross validations

cocl\_cross = cross\_validate(cocl, data, measures=['RMSE'], verbose=True)

# set metrics to display!

grid\_options=['KNNBasic', 'Funk SVD', 'NMF', 'SlopeOne', 'CoClustering']

test\_accs = [knnb\_cross['test\_rmse'].mean(), fsvd\_cross['test\_rmse'].mean(), nmf\_cross['test\_rmse'].mean(),

sone\_cross['test\_rmse'].mean(), cocl\_cross['test\_rmse'].mean()]

elapsed\_times = [sum(knnb\_cross['fit\_time']) + sum(knnb\_cross['test\_time']),

sum(fsvd\_cross['fit\_time']) + sum(fsvd\_cross['test\_time']),

sum(nmf\_cross['fit\_time']) + sum(nmf\_cross['test\_time']),

sum(sone\_cross['fit\_time']) + sum(sone\_cross['test\_time']),

sum(cocl\_cross['fit\_time']) + sum(cocl\_cross['test\_time'])]

# plot accuracy and time elapsed

fig, ax = plt.subplots(1, 2, figsize = (15, 8))

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

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

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

ax[0].set\_xlabel('Models')

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

ax[0].set\_title(f'Models versus RMSE Accuracy Score best @{grid\_options[np.argmax(test\_accs)]}')

ax[0].legend()

# time plot

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

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

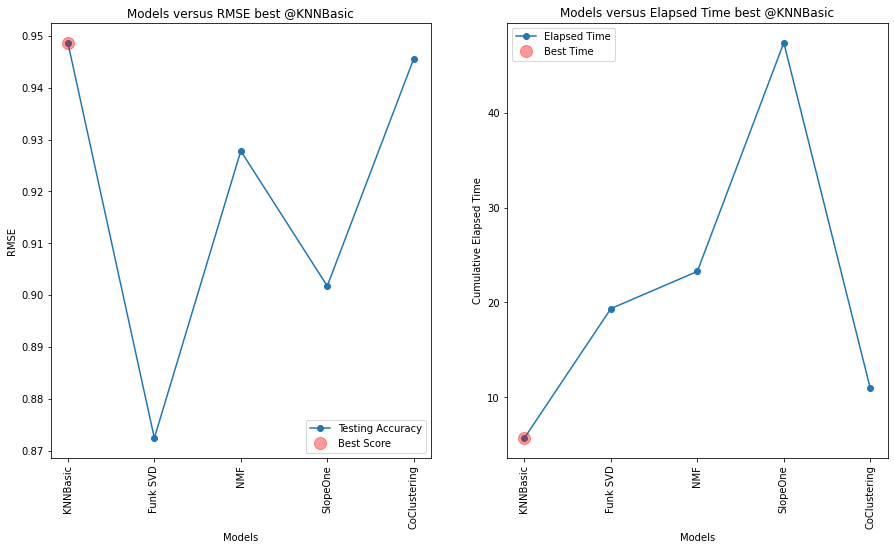
ax[1].set\_xlabel('Models')

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

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

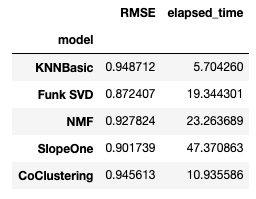
ax[1].plot(np.argmin(elapsed\_times), min(elapsed\_times), 'ro', markersize = 12, alpha = 0.4, label = 'Best Time')

ax[1].legend()



KNNBasic came with an impressive results, also it is the fastest.

|  |  |  |
| --- | --- | --- |
| **model** | **RMSE** | **elapsed\_time** |
| **KNNBasic** | 0.948712 | 5.704260 |
| **Funk SVD** | 0.872407 | 19.344301 |
| **NMF** | 0.927824 | 23.263689 |
| **SlopeOne** | 0.901739 | 47.370863 |
| **CoClustering** | 0.945613 | 10.935586 |



**Fine Tuning Models**

I fine tuned the model to improve above metrics. SlopeOne does not take any parameters, others take number of factors, epochs as well as regulation and biased, number of neighbors and clusters as hyperparameters.

sim\_params = {'name': 'cosine',

'user\_based': False # compute similarities between items

}

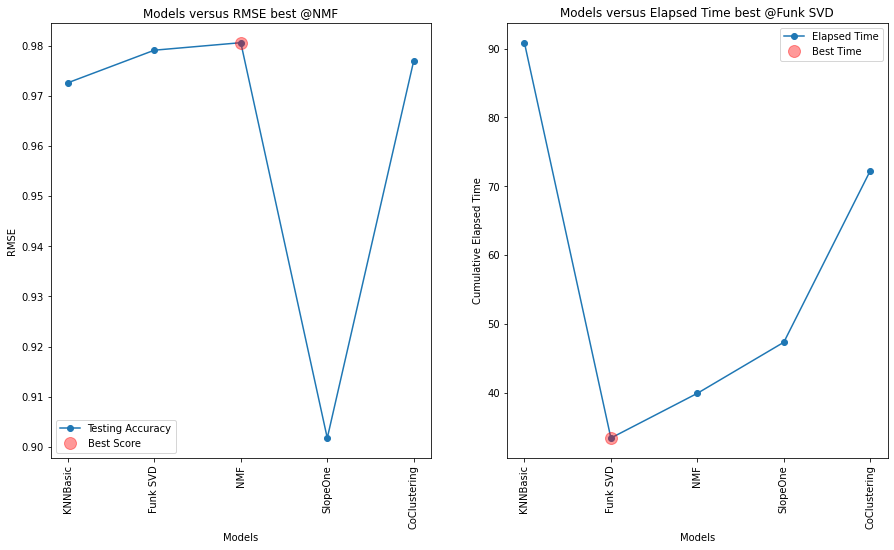
knnb2 = KNNBasic(random\_state = 93, k = 50, min\_k = 3, sim\_options = sim\_params)

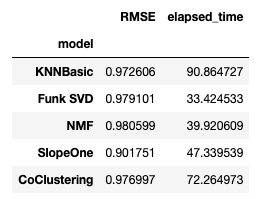
fsvd2 = SVD(random\_state = 93, n\_factors = 10, n\_epochs = 100, biased = False, lr\_all = 0.015)

nmf2 = NMF(random\_state = 93, n\_factors = 10, n\_epochs = 100, biased = False, reg\_pu = 0.022, reg\_qi = 0.022)

sone2 = SlopeOne()

cocl2 = CoClustering(random\_state = 93, n\_cltr\_u = 12, n\_cltr\_i = 12, n\_epochs = 100)





**Conclusion**

SlopeOne cannot be tuned, and it is the slowest algorithm out of the box, hence very easy to use. Remaining all 4 models can be tuned KNNBasic, Funk SVD, NMF, and CoClustering but they become slower with more iterations by n\_epoch. When similarity parameter switched to ‘cosine’ from ‘MSD’ on *KNNBasic* slowed down drastically like 16x, lost its best performing crown. Also, worth to note Funk SVD and NMF prone to overfit when results in *RMSE (testset)* seen greater than one, Funk SVD is the fastest in the fine tuning round.

I think KNNBasic and CoClustering are slightly easier to fine tune versus Funk SVD and NMF, however, *KNNBasic* is the slowest of all while fine tuning but performed the best out of the box both in RMSE and elapsed time metrics. Another observation, *KNNBasic* slows down with increase in number of distinct users. So, SlopeOne is easy to use followed by KNNBasic and CoClustering. Funk SVD and NMF are harder to tune.

RMSE can be higher than 1 that means score difference is more than 1, skews the results!

**Discussion 19.1: A 'Surprising' Recommendation - Section B**

I found a surprising recommendation in Amazon application. This *fuel treatment* item popped up out of blue in “You might also like” section. I have never purchased nor searched for it but I purchased some *air filters* for my cars and my cars are registered in the *garage*. However, the last car-related item purchase was like a year ago.



My guess is I looked up some car wax products on Amazon 6 months ago but did not purchase them. This might have triggered some models in *similar items* category based on what others people’s behavior by either purchasing or searching for. I typically purchase such flammable or liquid items at a local store, again it is not my interest to purchase *fuel treatment* online. I may fit to some customer profile who typically purchase them, this may be one culprit. Secondly, my customer profile information might have sold to Amazon as an outside factor via third-party data providers like The **Dun & Bradstreet** as every person has *dnb* id nowadays. I think third-party data services is the biggest influencer on online business hence in those recommendation systems besides in-house data.

 ————— o —————

**Module 20**

**Capstone Project**

**Notes:**

<https://jckantor.github.io/CBE30338/A.03-Animation-in-Jupyter-Notebooks.html> for Plot Animation!

All 4 models Logistic Regression, k-Nearest Neighbors, Decision Tree and Support Vector Machines were built and fed into GridSearchCV by using roc\_auc in scoring hyperparameter since it is binary classification per manual: “A receiver operating characteristic (ROC), or simply ROC curve, is a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied. It is created by plotting the fraction of true positives out of the positives (TPR = true positive rate) vs. the fraction of false positives out of the negatives (FPR = false positive rate), at various threshold settings. TPR is also known as sensitivity, and FPR is one minus the specificity or true negative rate.”

Each model fed with hyperparameter list to evaluate best outcome, they are captured in a table.

**A confusion matrix**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Predicted |  |
|  |  | **+** | **−** |
| **Actual** | **+** | TP | FN  Type II error |
|  | **−** | FP  Type I error | TN |

**Accuracy**

Accuracy is the most intuitive measure of performance, as it is simply the ratio of correctly predicted observations to total observations. Accuracy can be deceiving in that it may signal a highly accurate model, but in all actuality, it has some weaknesses. Accuracy is only useful when the dataset is perfectly symmetrical, where values of false negatives and false positives are almost identical with similar costs.

**Precision**

Precision is the proportion of accurately predicted positive observations in relation to the total predicted positive observations. High precision is directly correlated to a low false-positive rate.

**Recall**

Recall (a.k.a. sensitivity) is the proportion of correctly predicted positive observations in relation to all of the observations in an actual class. As a result, recall measures the precision with which our model can determine the relevant data.

**F1**

F1 is the weighted average of both precision and recall.

A table outlining the metrics that are typically used to determine the performance of a model

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **When to Use** | **Formula** | **Example** |
| **Accuracy** | Used when you have a perfectly symmetrical dataset | (TP + TN) ÷ (TP + FP + FN + TN) | One out of every ten labels is incorrect, and nine are correct. Therefore, the accuracy is 0.90. |
| **Precision** | Used when you want to be more confident of your true positives | TP ÷ (TP + FP) | Two out of every ten cancer samples labeled by our program are healthy, and eight are cancerous. Therefore, the precision is 0.80. |
| **Recall** | Used when the idea of false positives is far better than false negatives | TP ÷ (TP + FN) | Three out of every ten COVID-19 patients are mislabeled by our program as negative, and seven are labeled as positive. Therefore, the recall is 0.70. |
| **F1** | Used when you have uneven class distribution | 2 × (Recall × Precision) ÷ (Recall + Precision) | Four out of every ten healthy people are mislabeled as having COVID-19, and six are correctly labeled as healthy. Therefore, the recall is 0.60. |

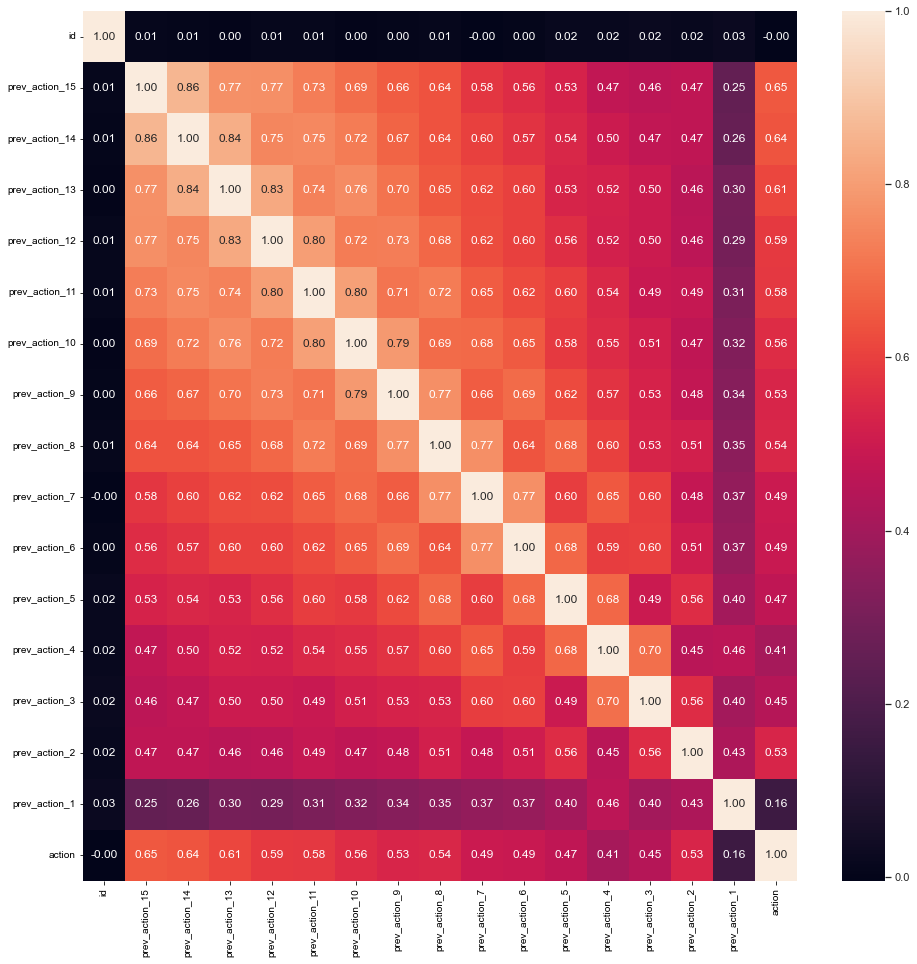
Example: false positive (classified malignant but benign) or false negative (classified as benign but malignant)

**Module Issues:**

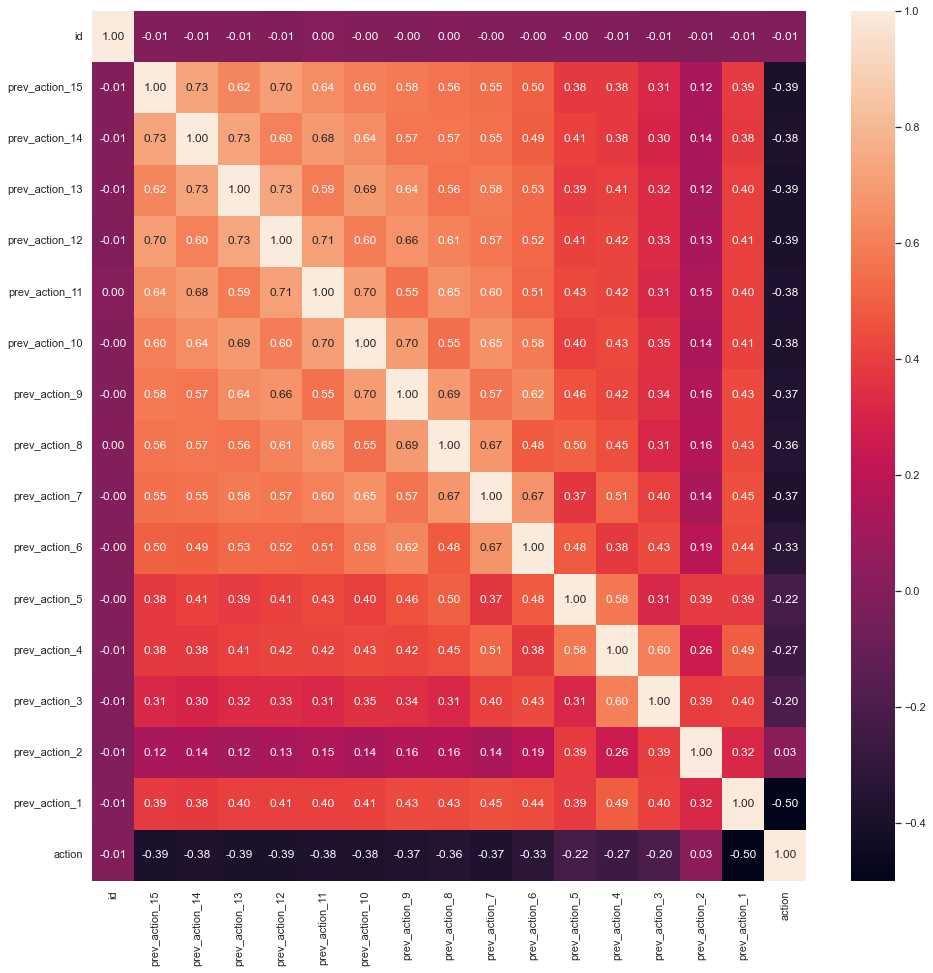
**Quizes:**

**Capstone**

**#1**

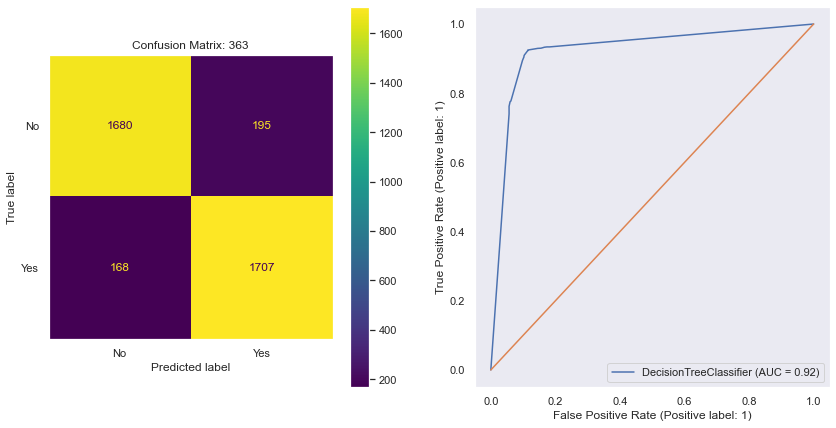


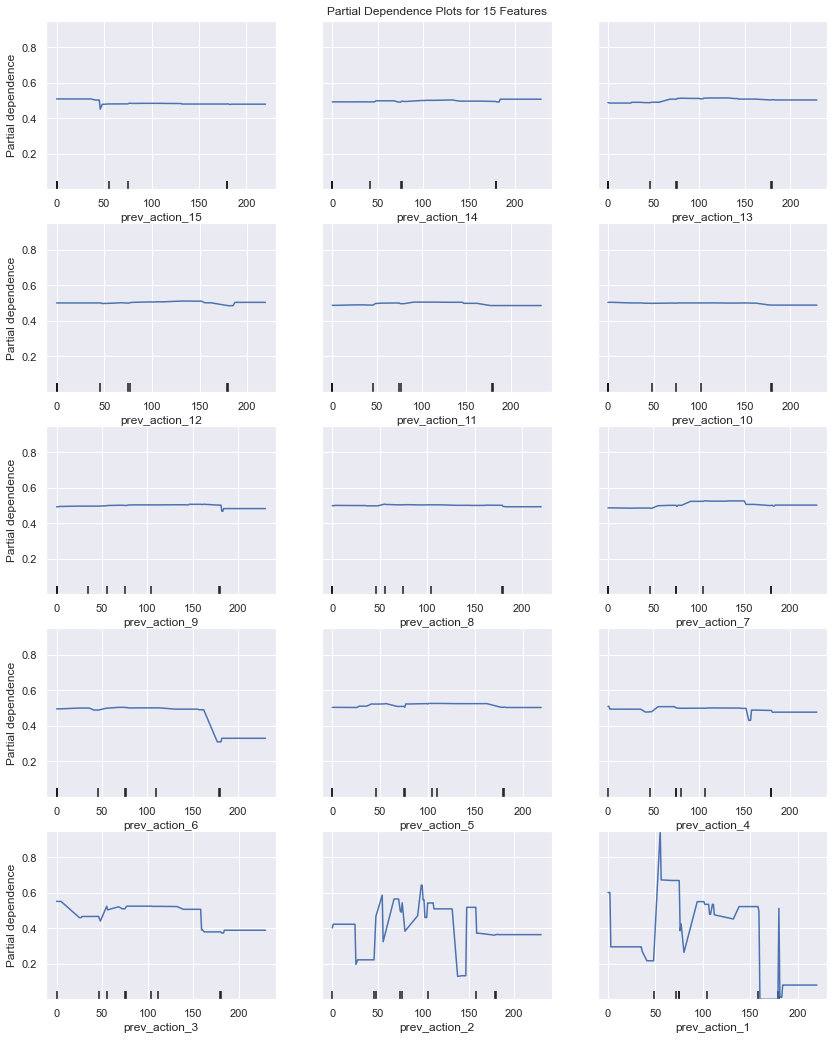
#2



Decision Tree

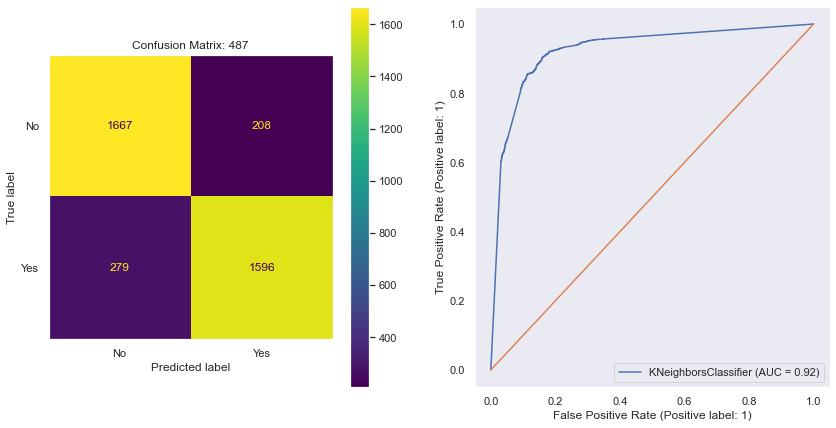
GridSearchCV came out with accuracy: {'criterion': 'entropy', 'max\_depth': 18, 'min\_samples\_leaf': 1, 'min\_samples\_split': 3, 'random\_state': 93} on the first dataset and roc\_auc: {'criterion': 'gini', 'max\_depth': 7, 'min\_samples\_leaf': 3, 'min\_samples\_split': 2, 'random\_state': 93} on the second dataset which has the lowest misclassifcation ratio of all. However, the hyperparameters on the second dataset caused more false positives. Running a further analysis by `{‘criterion': 'entropy', 'max\_depth': 18, 'min\_samples\_leaf': 1, 'min\_samples\_split': 3, 'random\_state': 93}` hyperparameters below:

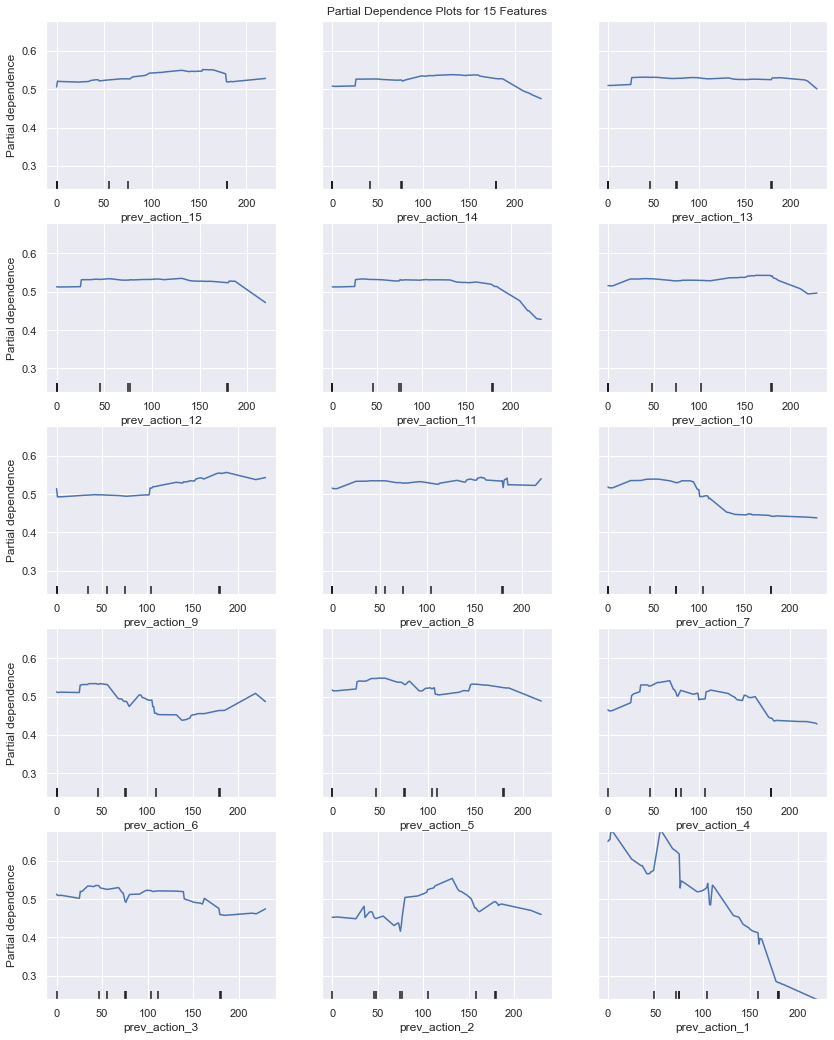




k-Nearest Neighbors

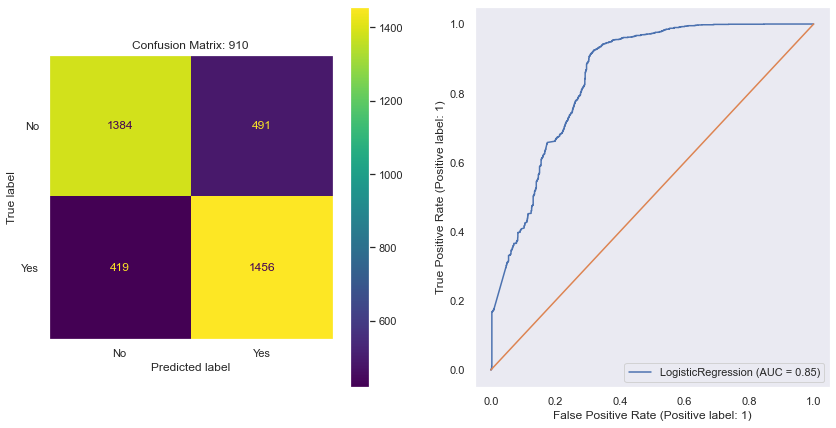
GridSearchCV came out with accuracy model with parameters {'n\_neighbors': 4, 'weights': 'distance'} on the first dataset and accuracy: {'n\_neighbors': 12, 'weights': 'distance'} on the second dataset which has the lowest misclassification ratio of all. Running the further analysis by `{’n\_neighbors': 4, 'weights': 'distance’}` hyperparameters below:

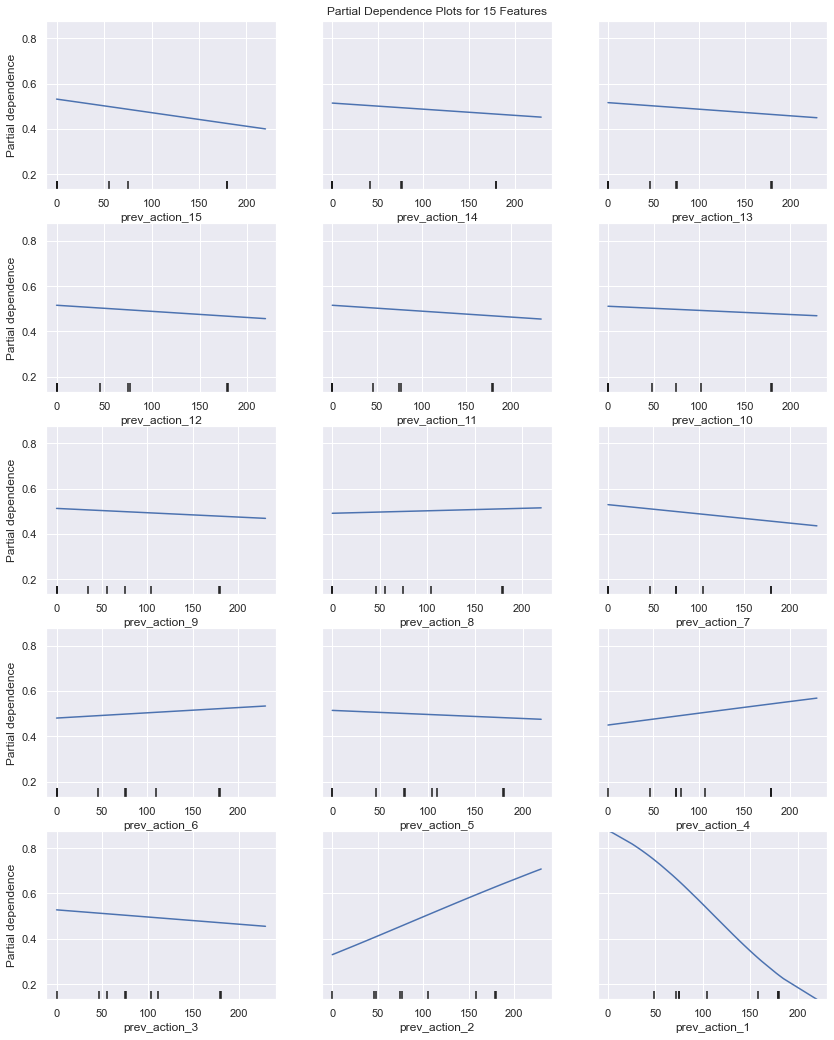




Logistic Regression

GridSearchCV came out with {'C': 0.0001, 'penalty': 'l2', 'random\_state': 93, 'solver': 'lbfgs'} on the first dataset and {'C': 0.0001, 'penalty': 'l2', 'random\_state': 93, 'solver': 'lbfgs'} on the second dataset which has the lowest misclassification ratio of all. Running the further analysis by `{‘C’: 0.0001, 'penalty': 'l2', 'random\_state': 93, 'solver': 'lbfgs’}` hyperparameters below:

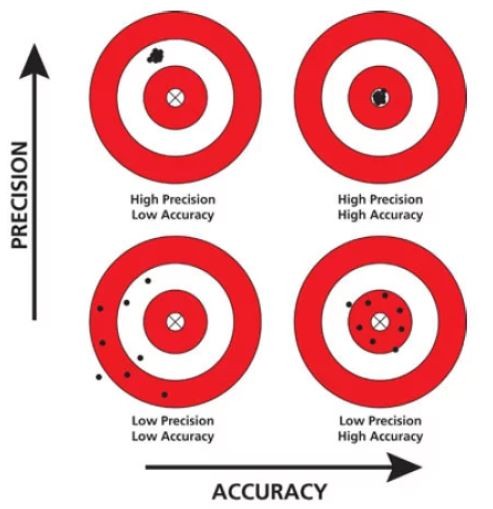




## **Next Steps**

Those models will highlight negative customer experiences, so, `false positive` rate should be low but skipping detection of negative experience (`false negative`) is also important considering proactive outreach to customer met not be desired on false positive cases. So, the model should minimize misclassifications therefore `accuracy` should be high as well as `precision`.

The decision tree model is outperforming other models and it is not sensitive to multicollinearity, I have not checked how sensitive to multicollinearity other models are, I will remove multicollinearity in the datasets and try those models as next steps. Decision tree is slow to train but k-Nearest Neighbors model is slow on execution, Logistic Regression model is the fastest but worst performer in this round.



Try **Random Forest** to implement a model with better accuracy and precision as Decision Trees, k-Nearest Neighbors and Support Vector Machines high variance and low bias!

<https://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning/>

**Bias Error**

Bias are the simplifying assumptions made by a model to make the target function easier to learn.

Generally, linear algorithms have a high bias making them fast to learn and easier to understand but generally less flexible. In turn, they have lower predictive performance on complex problems that fail to meet the simplifying assumptions of the algorithms bias.

1. **Low Bias**: Suggests less assumptions about the form of the target function.
2. **High-Bias**: Suggests more assumptions about the form of the target function.

Examples of **low-bias** machine learning algorithms include: Decision Trees, k-Nearest Neighbors and [Support Vector Machines](https://machinelearningmastery.com/support-vector-machines-for-machine-learning/).

Examples of **high-bias** machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

**Variance Error**

Variance is the amount that the estimate of the target function will change if different training data was used.

The target function is estimated from the training data by a machine learning algorithm, so we should expect the algorithm to have some variance. Ideally, it should not change too much from one training dataset to the next, meaning that the algorithm is good at picking out the hidden underlying mapping between the inputs and the output variables.

Machine learning algorithms that have a high variance are strongly influenced by the specifics of the training data. This means that the specifics of the training have influences the number and types of parameters used to characterize the mapping function.

* **Low Variance**: Suggests small changes to the estimate of the target function with changes to the training dataset.
* **High Variance**: Suggests large changes to the estimate of the target function with changes to the training dataset.

Generally, nonlinear machine learning algorithms that have a lot of flexibility have a high variance. For example, decision trees have a high variance, that is even higher if the trees are not pruned before use.

Examples of **low-variance** machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

Examples of **high-variance** machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.

**Bias-Variance Trade-Off**

The goal of any supervised machine learning algorithm is to achieve low bias and low variance. In turn the algorithm should achieve good prediction performance.

You can see a general trend in the examples above:

* **Linear** machine learning algorithms often have a high bias but a low variance.
* **Nonlinear** machine learning algorithms often have a low bias but a high variance.

The parameterization of machine learning algorithms is often a battle to balance out bias and variance.

Below are two examples of configuring the bias-variance trade-off for specific algorithms:

* The k-nearest neighbors algorithm has low bias and high variance, but the trade-off can be changed by increasing the value of k which increases the number of neighbors that contribute t the prediction and in turn increases the bias of the model.
* The support vector machine algorithm has low bias and high variance, but the trade-off can be changed by increasing the C parameter that influences the number of violations of the margin allowed in the training data which increases the bias but decreases the variance.

There is no escaping the relationship between bias and variance in machine learning.

* Increasing the bias will decrease the variance.
* Increasing the variance will decrease the bias.

There is a trade-off at play between these two concerns and the algorithms you choose and the way you choose to configure them are finding different balances in this trade-off for your problem

In reality, we cannot calculate the real bias and variance error terms because we do not know the actual underlying target function. Nevertheless, as a framework, bias and variance provide the tools to understand the behavior of machine learning algorithms in the pursuit of predictive performance.

 ————— o —————



**Module 21**

**Ensemble Techniques**

* [Video Transcripts](https://student.emeritus.org/courses/4765/files/3607249?wrap=1)
* [Download Video Transcripts](https://student.emeritus.org/courses/4765/files/3607249/download?download_frd=1)
* [Quick Reference Guide](https://student.emeritus.org/courses/4765/files/3607251?wrap=1)

**Glossary**

**AdaBoost**

Stands for adaptive boosting and is a technique that turns several weak classifiers into strong ones

**Bagging**

Fitting a number of decision trees on samples of the same dataset and averaging their predictions

**Boosting**

Adding ensemble members sequentially to correct the predictions made by previous models and producing a weighted average of the predictions

**Bootstrapping**

The process of drawing several samples from a single dataset using replacement

**Decision Stump**

A decision tree that uses only a single attribute for splitting

**Wisdom of the Crowd**

An idea that assumes large crowds are collectively smarter than individual experts; applied to ML/AI, the assumption is that many models will make better predictions than a single model

**Notes:**

Ensemble models combine multiple algorithms to improve the predictive performance of each algorithm individually. Ensemble models typically consist of two strategies—bagging and boosting—and there are many examples of predefined ensemble algorithms.

Bootstrap aggregation, or bagging, is a meta-learning technique in which many classifiers are trained on different partitions of the training data, and the resultant predictions of each of those classifiers are combined to make a final prediction.

**Bootstrapping and Bagging**

If our base models have high **variance**, then **bagging** will be the solution. If they are highly **biased**, then **boosting** will help.

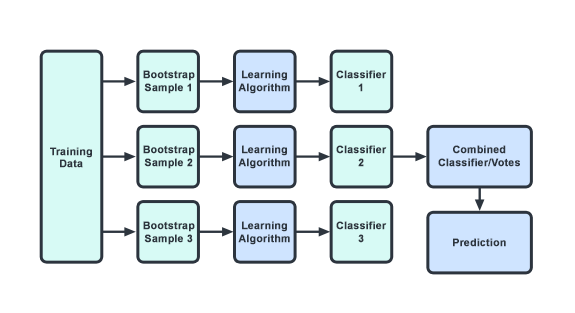
**Bagging Classifiers and Bagging Regressors**

Bagging classifiers and regressors are ensemble meta-estimators that fit the regressor and classifier models to random subsets of the original dataset. A final prediction is created by combining the predictions from each model. In these meta-estimators, randomization is introduced into the model-building process. Finally, the outcomes are aggregated to reach a categorical outcome: the aggregation averages over the iterations for a numerical target variable.

**Bagging Classifiers**

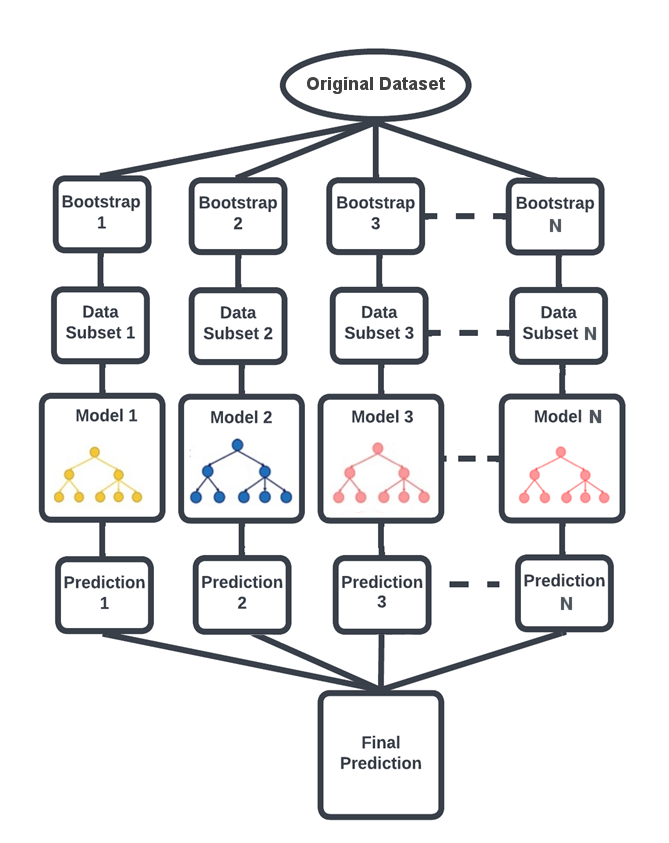
Bagging classifiers aggregate the individual predictions of base classifiers on a random subset of the original dataset (either by voting or by averaging) to form a final prediction. An ensemble of this type of meta-estimator can be calculated by introducing randomization into the construction procedure of a black-box estimator and then reducing its variance.

In the image below the training data is split into three bootstrapped samples. Each sample is sent to a separate alrgorithm. The output of each algorithm is then fed to their own classifier. The results of each of the classifiers are then aggregated to arrive at a final prediction.



**Bagging Regressors**

Bagging regressors are similar to bagging classifiers. Each regressor model is trained on a random subset of the original training set and the predictions are aggregated. Since the target variable is numeric, the aggregation averages over iterations.



**Random Forests**

Random forests are ensemble techniques that include multiple decision trees and a method called bootstrap and aggregation that is commonly referred to as bagging and which can perform both regression and classification tasks. This method combines multiple decision trees into one final output rather than relying on individual decision trees.

**Boosting**

Boosting is an ensemble learning technique that lowers the number of errors by combining weak learners into one strong learner. Boosting involves selecting a random sample of data, fitting a model, and then training each model sequentially, that is, each model tries to compensate for the weaknesses of the one before it. Boosting algorithms might differ in how they create weak learners and aggregate them during sequential reinforcement. Here are three popular types:

* Adaptive Boosting (AdaBoost)
* Gradient Boosting
* Extreme Gradient Boosting (XGBoost)

**Module Wrap-Up**

The objective of an ensemble of models is to combine multiple models so that the resulting prediction is the best possible one, based on all predictions. Ensemble methods are ideal for reducing the variance in models and thus increasing the accuracy of predictions. Notable ensemble techniques include boosting and bagging.

import warnings

warnings.filterwarnings('ignore')

**Module Issues:**

**All Codio 21 Activities:** Not accessible:



**Codio 21.1 Problem 2:** no description

**Codio 21.1 Problem 3:** description is not clear!

**Codio 21.1 Problem 5 & 6:** description is not clear to include SVC(probability = True) and the instructions is misleading for weighted\_acc in Problem 6.

**Codio 21.2 Problem 2:** no description

**Codio 21.3 Problem 2:** variable is supposed to be coef\_df

**Codio 21.7 Problem 2:** assert boost\_acc == boost\_acc\_ failing, although, they are the same! The runs produce different results: 0.9492481203007519, 0.9511278195488722 although the latter is right answer. The workaround is do it twice and use the result from the second model, make sure you get 0.9511278195488722!

**Quizes:**

What type of function takes a sample and returns a prediction? : Predictor, Classifier

*You are correct! The answers “*Predictor*” and “*Classifier*” are correct because a predictor or a classifier is a function that takes a sample and returns a prediction for that sample*

Which of the following is an example of bagging? : Random tree

*You are correct! The answer “*Random tree*” is correct because this is an example of bagging.*

The formula for ensemble variance is

Variance[Ensemble] = N/Variance[Individual] : False

*You are correct! The answer “*False*” is correct because the formula for ensemble variance is*

*“*Variance[Ensemble] = Variance[Individual]/N.”

What is the important condition for trusting the wisdom of the crowd? : The decisions are made independently

*You are correct! The answer “*The decisions are made independently*” is correct because this is the important condition for trusting the wisdom of the crowd.*

In ensemble learning, the training data is trained on a single model. : False

*You are correct! The answer “*False*” is correct because in ensemble learning the training data is trained on many models.*

In the aggregation step of the metamodel, what is hard voting? : The selection of a prediction based on the class that receives the most votes

*You are correct! The answer “*The selection of a prediction based on the class that receives the most votes*” is correct because in hard voting* *the class that receives the most votes for a particular test point is selected as the prediction for that test point.*

In ensemble learning for regression problems, the output of the models is combined by simple averaging. : True

*You are correct! The answer “*True*” is correct because for regression problems the aggregate step in the metamodel is a simple average.*

The metamodel can also assign weights(blank) to the individual models and thus give each model a greater or smaller influence in the final decision. : α1 through αm

*You are correct! The answer “*α1 through αm*” is correct because the weights for models M1 through Mm are represented as “*α1, α2, ..., αm.”

What function in Python is used for majority vote? : Mode()

*You are correct! The answer “*Mode()*” is correct because the majority vote is the selection of a prediction based on the class that receives the most votes. It is equivalent to mode functionality.*

In ensemble learning, if the base models have high variance, the solution is boosting. : False

*You are correct! The answer “*False*” is correct because if the base models have high variance, the solution is bagging.*

Bootstrap sampling is (blank). : With replacement

*You are correct! The answer “*With replacement*” is correct because bootstrap sampling has to be done with replacement.*

What is the chance of an item being included at least once in the bootstrap sample in N trials? : 1−(1−1/N)^N

*You are correct! The answer “*1−(1−1/N)N*” is correct because this is the chance of an item being selected at least once in the bootstrap sample.*

The bootstrap sample taken from a training dataset D = {1, 2, 3, 4, 5} is D1 = {2, 3, 2, 4, 3}.

Which of the following is the out-of-bag sample? : {1, 5}

*You are correct! The answer “*{1, 5}*” is correct because these are the items from the training data that are not the part of the bootstrap sample; hence they are called out-of-bag samples.*

What is the constructor used in the Python function BaggingClassifier()to request the out-of-bag score? : oob\_score = True

*You are correct! The answer “*oob\_score = True*” is correct because this is the constructor used to request the out-of-bag score.*

A random forest is an algorithm that increases the correlation between trees in a bagged ensemble by introducing randomness into the training process. : False

*You are correct! The answer “*False*” is correct because a random forest is an algorithm that reduces the correlation between trees in a bagged ensemble.*

In scikit-learn, which parameter tells the algorithm to design each split based on a randomly chosen set of two features? : max\_features = 2

*You are correct! The answer “*max\_features = 2*” is correct because this parameter tells the algorithm to design each split based on a randomly chosen set with a maximum of two features.*

Which of the following is the correct statement for importing the random forest classifier from scikit-learn? : from sklearn.ensemble import RandomForestClassifier

*You are correct! The answer “*from sklearn.ensemble import RandomForestClassifier*” is correct because this is the correct statement for importing random forest classifiers from the Python library scikit-learn.*

The parameter “n\_estimators” in the function “RandomForestClassifier()” is used to declare the number of trees in the forest. : True

*You are correct! The answer “*True*” is correct because this parameter is used to declare the number of trees to be built in the forest.*

Which of the following are the correct steps to measure the relative importance of a feature in a random forest? : Step 1: Finding all of the nodes in the forest that split along that feature, Step 2: Adding up the reductions in entropy that they produced, weighted by the number of data points in each node

*You are correct! The answers “*Step 1: Finding all of the nodes in the forest that split along that feature,*” and “*Step 2: Adding up the reductions in entropy that they produced, weighted by the number of data points in each node*” are correct because these are the steps to measure the relative importance of a feature in a random forest.*

What do you call a tree with only one node that slices the dataset in alignment with one of the features? : Decision stump

*You are correct! The answer “*Decision stump*” is correct because a tree with only one node that cuts the dataset with a slice aligned with one of the features is called a decision stump.*

The equation for stump misclassification score is denoted as ϵs. : ϵs = Σmisclass Wsi

*You are correct! The answer “*ϵ*s = Σmisclass Wsi” is correct because this is the correct equation for stump misclassification score.*

The influence parameter αs is used to update the weights for each sample. If the sample was correctly classified, then what will its weight be equal to? : Wsi e-αs

*You are correct! The answer “Wsi e-αs” is correct because if the sample was correctly classified then its weight is divided by e to the power*αs*, which causes it to decrease.*

AdaBoost boosting algorithms are not easily overfitted. : True

*You are correct! The answer “*True*” is correct because AdaBoost is very forgiving and gives plenty of time to stop the training process before reaching the point of overfitting, making these algorithms slow learners.*

This statement in Python is for building an AdaBoost model with a decision tree as the base model:

Model = AdaBoostClassifier(DecisionTreeClassifier) : False

*You are correct! The answer “*False*” is correct because the statement in Python for building* *an AdaBoost model with a decision tree as the base model is “*Model = AdaBoostClassifier(DecisionTreeClassifier(max\_depth))*.”*

Gradient boosting refers to the idea of applying gradient descent to the problem of boosting. : True

*You are correct! The answer “*True*” is correct because gradient boosting means applying gradient descent to boosting problems.*

What is the cost function for gradient boosting trees? : Squared loss

*You are correct! The answer “*Squared loss*” is correct because the cost function for gradient boosting trees is squared loss.*

In the gradient boosting tree algorithm, what is the step added in the model “H” toward the desired direction “ri”? : αh (alpha)

*You are correct! The answer “*αh*” is correct because the step added in the model “H” toward the desired direction “ri” is “H* ← *H +*αh.”

Which of the following are properties of boosting algorithms? : Does not increase variance as much as other algorithms, Reduces the bias of weak learners

*You are correct! The answers “*Reduces the bias of weak learners*” and “*Does not increase variance as much as other algorithms*” are correct because these are the properties of boosting algorithms.*

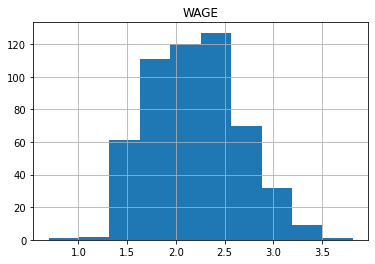
**Try-It Activity 21.1: Comparing Aggregate Models for Regression - Section B**

**Exploratory Data Analysis**

The dataset contains categorical and numerical variables, the data contains wage and demographic information on 534 individuals, it has no missing values in features. The dependent variable as real number in dollars per hour is right-skewed that needs log() function to normalize.

np.log1p(y).hist()

plt.show()



Histogram of all features after encoding:

# tranform features and plot histogram

pd.merge(

survey.select\_dtypes(include=np.number),

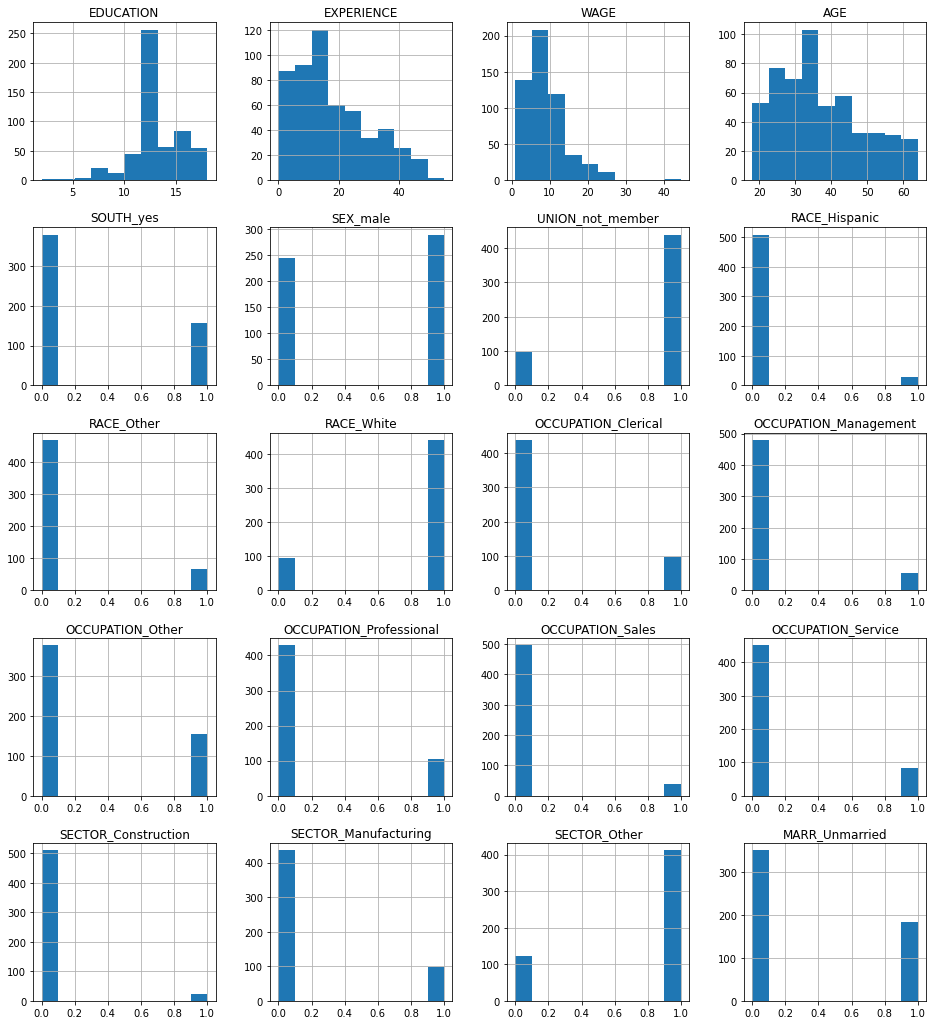
pd.DataFrame(ohe.fit\_transform(X.select\_dtypes(include = 'category')),

columns=ohe.get\_feature\_names\_out()),

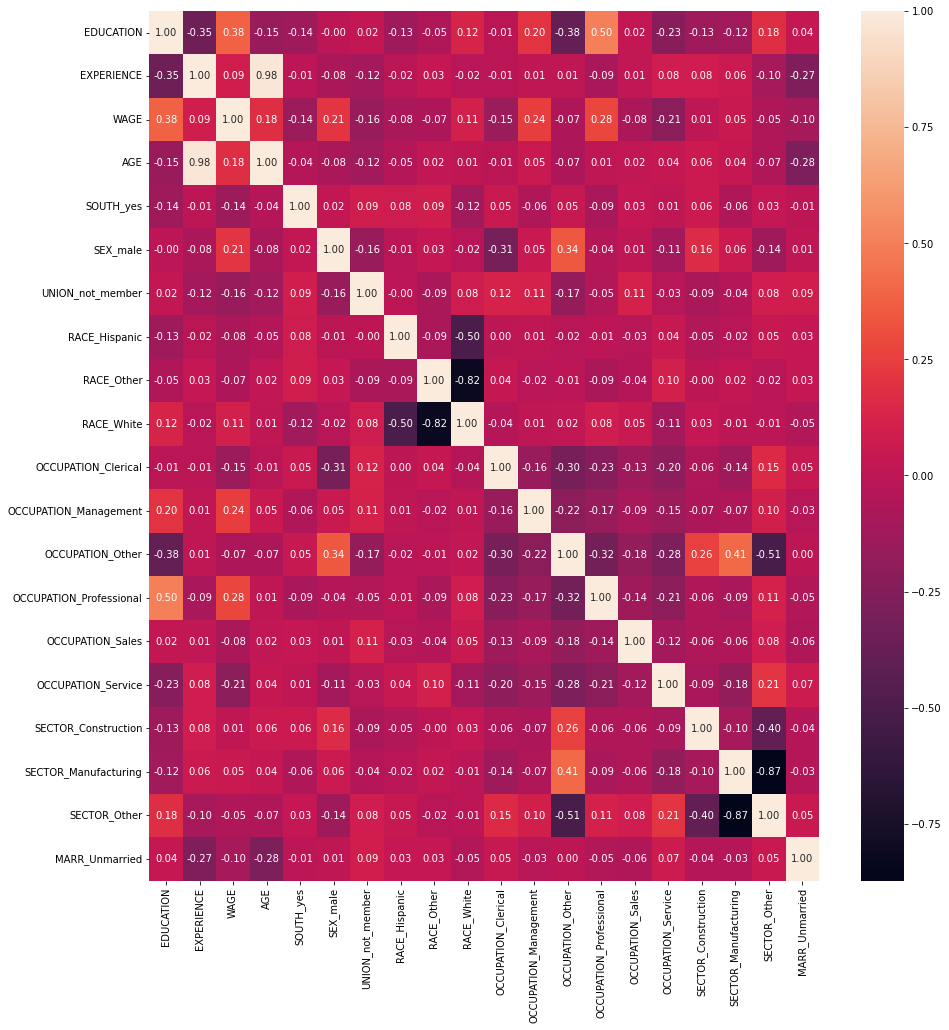
left\_index=True, right\_index=True

).hist(figsize = (16, 18))

plt.show()

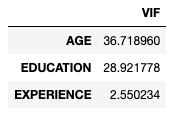


The correlation map shows a string correlation between EXPERIENCE and AGE, something to consider in the feature engineering.



**Feature Engineering**

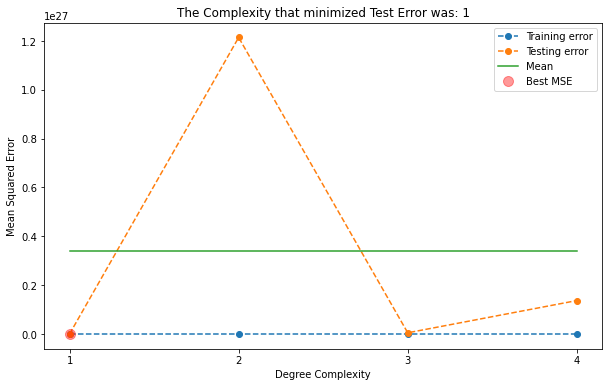
Variance Inflation Factor (VIF) on numerical features points two features as multicollinearity as below:



As the correlation map also pointed AGE, I verified removing this column helped stabilizing LinearRegression and Ridge models in my tests with or without this feature.

**Building Models**

First I checked if higher degree models make any difference 1 through 4, degree = 1 performed well in this analysis:



I ran all 5 models without any PolynomialFeatures:

* LinearRegression with TransformedTargetRegressor
* KNeighborsRegressor
* DecisionTreeRegressor
* Ridge with TransformedTargetRegressor
* SVR

# Encoder and Scaler Step

features = X.select\_dtypes(include = 'category').columns

ohe\_step = make\_column\_transformer( (OneHotEncoder(drop = 'if\_binary'), features),

(StandardScaler(), X.select\_dtypes(include=np.number).columns) )

And 5 models:

ttrl\_pipe = Pipeline([('transformer', ohe\_step),

('ttregressor', TransformedTargetRegressor(func = np.log1p,

inverse\_func = np.expm1,

regressor = LinearRegression(fit\_intercept=False))) ])

#fit on train

ttrl\_pipe.fit(X\_train, y\_train)

ttrl\_train\_acc = ttrl\_pipe.score(X\_train, y\_train)

ttrl\_test\_acc = ttrl\_pipe.score(X\_test, y\_test)

ttrl\_test\_mse = mean\_squared\_error(y\_test, ttrl\_pipe.predict(X\_test))

knn\_pipe = Pipeline([('transformer', ohe\_step),

('model', KNeighborsRegressor())

])

knn\_param\_dict = {'model\_\_n\_neighbors': [9,10,12,14,16,18], 'model\_\_weights': ['uniform', 'distance']}

knn\_grid = GridSearchCV(knn\_pipe, param\_grid = knn\_param\_dict)

#fit on train

knn\_grid.fit(X\_train, y\_train)

knn\_train\_acc = knn\_grid.best\_estimator\_.score(X\_train, y\_train)

knn\_test\_acc = knn\_grid.best\_estimator\_.score(X\_test, y\_test)

knn\_test\_mse = mean\_squared\_error(y\_test, knn\_grid.best\_estimator\_.predict(X\_test))

svr\_pipe = Pipeline([('transformer', ohe\_step),

('model', SVR())

])

svr\_param\_dict = {'model\_\_kernel': ['linear', 'rbf', 'sigmoid', 'poly']}

svr\_grid = GridSearchCV(svr\_pipe, param\_grid = svr\_param\_dict)

#fit on train

svr\_grid.fit(X\_train, y\_train)

svr\_train\_acc = svr\_grid.best\_estimator\_.score(X\_train, y\_train)

svr\_test\_acc = svr\_grid.best\_estimator\_.score(X\_test, y\_test)

svr\_test\_mse = mean\_squared\_error(y\_test, svr\_grid.best\_estimator\_.predict(X\_test))

dtr\_pipe = Pipeline([('transformer', ohe\_step),

('model', DecisionTreeRegressor())

])

dtr\_param\_dict = {'model\_\_max\_depth': [1,2,3,4,5,6,7,8,9,10,12,14,16,18], 'model\_\_splitter':['best', 'random']}

dtr\_grid = GridSearchCV(dtr\_pipe, param\_grid = dtr\_param\_dict)

#fit on train

dtr\_grid.fit(X\_train, y\_train)

dtr\_train\_acc = dtr\_grid.best\_estimator\_.score(X\_train, y\_train)

dtr\_test\_acc = dtr\_grid.best\_estimator\_.score(X\_test, y\_test)

dtr\_test\_mse = mean\_squared\_error(y\_test, dtr\_grid.best\_estimator\_.predict(X\_test))

ttrr\_pipe = Pipeline([('transformer', ohe\_step),

# ('scaler', StandardScaler()),

('ttregressor', TransformedTargetRegressor(func=np.log1p,

inverse\_func=np.expm1,

regressor=Ridge())) ])

ttrr\_param\_dict = {'ttregressor\_\_regressor\_\_alpha':

[0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 1.0, 10.0, 100.0, 1000.0]}

ttrr\_grid = GridSearchCV(ttrr\_pipe, param\_grid = ttrr\_param\_dict)

#fit on train

ttrr\_grid.fit(X\_train, y\_train)

ttrr\_train\_acc = ttrr\_grid.best\_estimator\_.score(X\_train, y\_train)

ttrr\_test\_acc = ttrr\_grid.best\_estimator\_.score(X\_test, y\_test)

ttrr\_test\_mse = mean\_squared\_error(y\_test, ttrr\_grid.best\_estimator\_.predict(X\_test))

**VotingRegressor Models**

I built a VotingRegressor which outperformed all the individual models:

voter1 = VotingRegressor(estimators=[('ttr', ttrl\_model ),

('knn', knn\_model ),

('svm', svr\_model ),

('dtr', dtr\_model ),

('rid', ttrr\_model )

],

verbose = True).fit(X\_train, y\_train)

vote1\_train\_acc = voter1.score(X\_train, y\_train)

vote1\_test\_acc = voter1.score(X\_test, y\_test)

vote1\_test\_mse = mean\_squared\_error(y\_test, voter1.predict(X\_test))

I also built a second VotingRegressor model by assigning weights per their individual performance which performed slightly better:

voter2 = VotingRegressor(estimators=[('ttr', ttrl\_model ),

('knn', knn\_model ),

('svm', svr\_model ),

('dtr', dtr\_model ),

('rid', ttrr\_model )

],

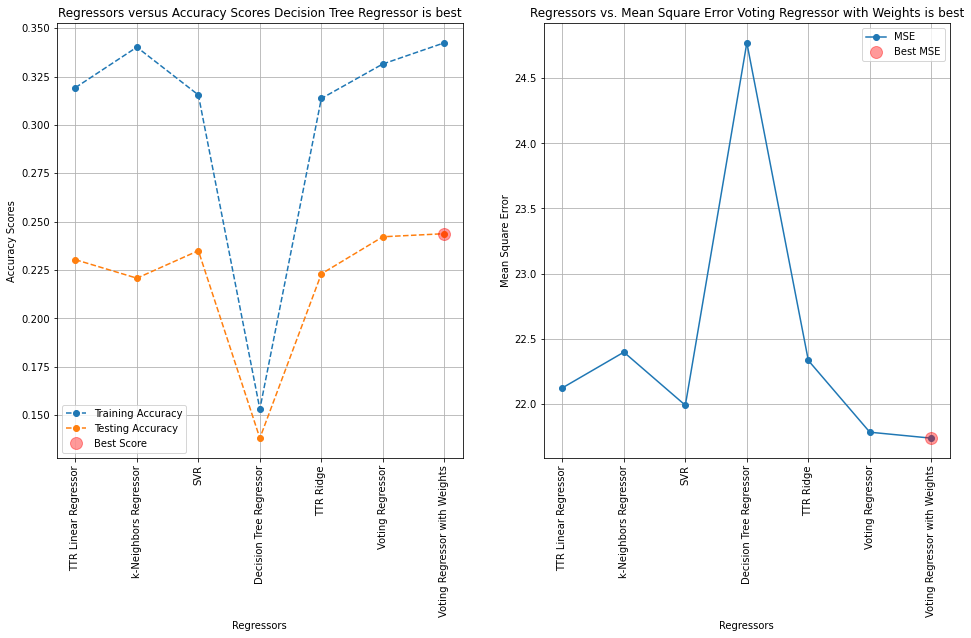
verbose = True, weights=[4,6,6,1,4]).fit(X\_train, y\_train)

vote2\_train\_acc = voter2.score(X\_train, y\_train)

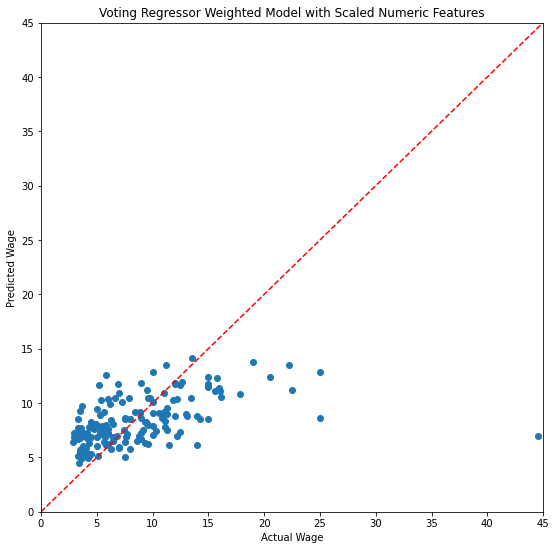
vote2\_test\_acc = voter2.score(X\_test, y\_test)

vote2\_test\_mse = mean\_squared\_error(y\_test, voter2.predict(X\_test))

The comparison plot shows weighted VotingRegressor model performed well has the least MSE and highest accuracy score:



I plotted actual versus predicted WAGE plot, the models perform well up to $15/hour wages, after then they are struggling to predict higher wage earners which is 10% of the population, somewhat outliers:



**Feature Importance**

Finally we can use the weighted VotingRegressor model on feature importance which shows what features have high influence on the prediction. Here are top 5 features:

* EDUCATION
* OCCUPATION
* EXPERIENCE
* SEX
* UNION

def column\_importance():

# fit model with training set

print('model r^2 :', voter2.score(X\_test, y\_test))

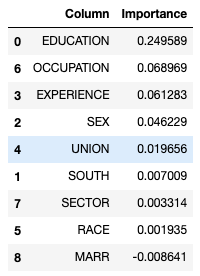
# permutation importance

r = permutation\_importance(voter2, X\_test, y\_test, n\_repeats = 50, random\_state = 93)

print('importance:', r.importances\_mean)

return pd.DataFrame({"Column":X.columns, "Importance":r.importances\_mean}).sort\_values(

by = "Importance", ascending = False)



fi.plot.barh(figsize=(9, 7))

plt.title('Feature Importance')

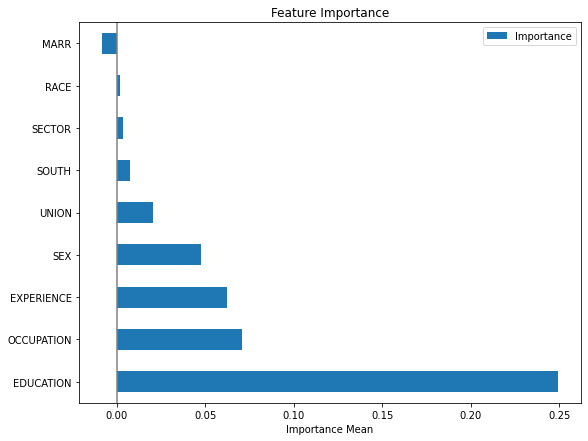
plt.xlabel('Importance Mean')

plt.axvline(x=0, color='.5')

plt.yticks(ticks=fi.reset\_index().sort\_values(by='index')['Column'].index,

labels=fi.reset\_index().sort\_values(by='index')['Column'])

plt.show()



**Conclusion**

Linear models are subject to multicollinearity, TransformedTargetRegressor log transformation on the dependent variable helped in LinearRegression and Ridge estimators, the weighted ensemble VotingRegressor model performed the best. Feature importance can be applied to this model even though the wisdom of the crowd technique adds complexity to the model understanding, but feature importance clarifies it.

**Next Steps**

Hyperparameters via GridSearchCV can be applied on ensemble VotingRegressor model and its estimators for further fine tuning. Besides, RandomForestRegressor and GradientBoostingRegressor should be tried along with other estimators. Also, the high earners in this dataset is underpopulated just 55 out of 534 so their predictions are skewed, this outlier case perhaps can be handled in a separate model.

 ————— o —————



**Module 22**

**Notes:**

**Module Issues:**

**Quizes:**

 ————— o —————

**Module 21**

**Notes:**

**Module Issues:**

**Quizes:**