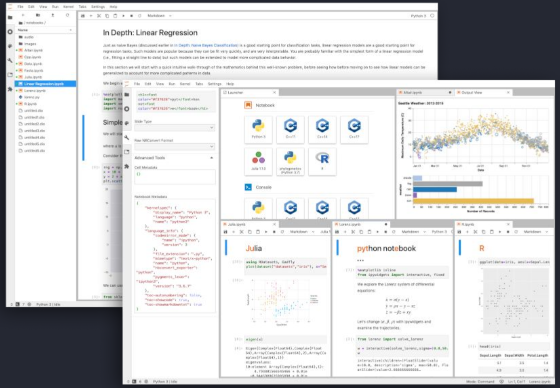
Classifying Applications, Structure, and Data Transformations in Jupyter Notebooks: An Exploratory Study

Jacob Yim and Qinshi Zhang



Research Questions

- **High-level question**: How do Jupyter Notebook users currently use notebooks?
- We break this topic into three research questions, focusing around **applications**, **structure**, and **data transformations**.
- We begin to address these questions using a **corpus study** of **30 real-world Jupyter Notebooks.**

Research Questions

- RQ1: What **types of tasks** do real users apply **Jupyter Notebooks** to?
- RQ2: How do users **structure** their notebooks to accomplish these tasks?
- RQ3: What **data transformations** do users perform in notebooks to accomplish these tasks?

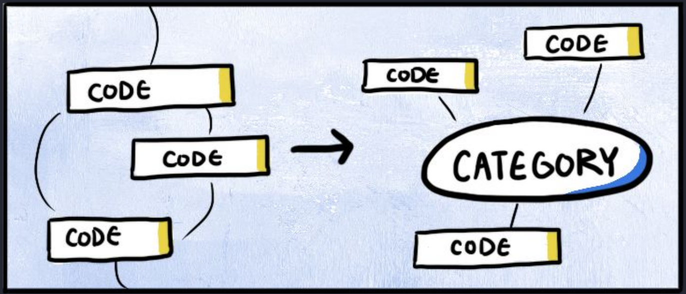
Research Impact

- Jupyter Notebooks are popular for working with data, with over 10M public notebooks on GitHub[1]
- By better understanding use cases for notebooks, we may be able to build tools that are more closely aligned with users' needs.
- Notebooks have been criticized for encouraging disorganized analyses
  - Understanding how notebook users currently structure their notebooks could lead to new insights
- **Data wrangling** is reportedly time-consuming and tedious
  - Data transformation may be a good target for automation
- This is an extremely preliminary study to inform future tool-building directions

1. https://cdss.berkeley.edu/news/project-jupyter-celebrates-20-years-fernando-perez-reflects-how-it-started-open-sciences

Procedure

- Conducted a **corpus study** using 30 publicly-available Jupyter Notebooks from GitHub, via the Hugging Face dataset The Stack[1]
- **Open coding**: manually assigned codes relating to research questions to sections of notebooks, building codebook iteratively (treating notebooks as qualitative data)
  - For each notebook: codes relating to its use case, structure, and data transformations
- **Content analysis**: informal quantitative analysis of codes



1. https://huggingface.co/datasets/bigcode/the-stack

Results

RQ1: What **types of tasks** do real users apply **Jupyter Notebooks** to?

We identified four broad categories of notebooks in our dataset:

- **Modeling**: feed data to a model, usually using a library, ostensibly to make predictions or perform analysis
- **Data Analysis**: show properties of an existing dataset and relationships between features
- **Scripting**: automate some process or transformation, typically saving transformed files
- **Educational**: demonstrate some concept using toy data, or an assignment notebook

These categories applied to all but one notebook in our sample (which was only a few cells long)

Results

RQ1: What **types of tasks** do real users apply **Jupyter Notebooks** to?

Category	**Count**
Modeling	11
Educational	11
Data Analysis	4
Scripting	3
Indeterminate	1

Results

RQ1: What **types of tasks** do real users apply **Jupyter Notebooks** to?

- Many more **educational** notebooks than expected (11/30 = ~36.7%)
  - Education-related notebooks may be more likely to be public on GitHub, and may be overrepresented since assignments are duplicated by students
- Fewer **data analysis** notebooks than expected (4/30 = ~13.3%)
  - May carry too much overlap with Modeling code
  - Users may not make repositories with personal data public
- **Scripting** is a surprising use of Jupyter Notebooks

Category	**Count**
Modeling	11
Educational	11
Data Analysis	4
Scripting	3
N/A	1

Results

RQ1: What **types of tasks** do real users apply **Jupyter Notebooks** to?

- In addition, we classified the **types of data** users were primarily working with in each notebook:
- Notebooks in the sample dealt with a wide variety of types of data, from text and numerical data to TIF and MRI files

Category	Count
Numerical	11
Text	8
Mixed	4
Other file types	3
N/A	4

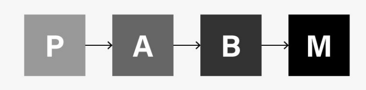
Results

RQ2: How do users **structure** their notebooks to accomplish these tasks?

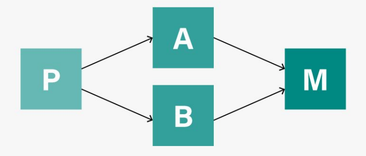
We identified two types of structure:

- **Linear**: Steps generally follow from one another sequentially and build upon one another
- **Parallel**: Incorporates some element of comparison between different approaches

These categories applied to all notebooks in our sample.



Serial Process



Parallel Process

Results

RQ2: How do users **structure** their notebooks to accomplish these tasks?

- A significant fraction of notebooks (8/30 = ~26.7%) incorporated structures that we loosely defined as “parallel”: comparing between different approaches or different variations of similar code
  - e.g. for comparing different variations of a model, or running the same model on different datasets
  - Many of these notebooks had cells sharing near-duplicate code

All notebooks

Category	Count
Linear	22
Parallel	8

Excluding “educational” notebooks

Category	Count
Linear	13
Parallel	6

```python
#Doc2Vec model
def doc2vec_evaluate(fullsetdf,subsetdf):
    size = 300
    window = 5
    min_count = 1
    workers = 4
    sg = 1
    df = preprocess(fullsetdf)
    df.drop(df.columns[[0]], axis=1, inplace=True)
    # Tokenize the text column to get the new column 'tokenized_text'
    df['tokenized_text'] = [simple_preprocess(line, deacc=True) for line in df['sentence']]
    documents = [TaggedDocument(doc, [i]) for i, doc in enumerate(df['tokenized_text'])]
    #Initialize the model
    doc2vec_model = Doc2Vec(documents, vector_size=size, window=window, min_count=min_count, workers=workers)
    for index, row in df.iterrows():
        model_vector = doc2vec_model.infer_vector(row['tokenized_text'])
        if index == 0:
            header = ",".join(str(ele) for ele in range(size))
            header = header.split(',')
            doc2vec_df = pd.DataFrame([], columns = header)
        #if type(model_vector) is list:
        line1 = ",".join( [str(vector_element) for vector_element in model_vector] )
        line1 = line1.split(',')
        #else:
        #    line1 = ",".join([str(0) for i in range(size)])
        a_series = pd.Series(line1, index = doc2vec_df.columns)
        doc2vec_df =  doc2vec_df.append(a_series,ignore_index=True)
    y_train = pd.DataFrame(fullsetdf['label'])
    X_train = doc2vec_df
    lr_clf = LogisticRegression()
    dt_clf = DecisionTreeClassifier()
    rf_clf = RandomForestClassifier()
    #ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
    nb_clf = GaussianNB()
    nn_clf = MLPClassifier(random_state=1, max_iter=300)
    svm_clf = svm.SVC(gamma=0.001, C=100.)
    scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
    print(" *** Full dataset doc2vec features *** ")
    scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    #scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    print("MODELS              ,Accuracy , Precision  ,  Recall    ,  F-score            ", flush=True)
    print("Logistic Regression ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].mea
    print("Decision Tree       ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].mea
    print("Random Forest       ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].mea
    #print("Adaboost            ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].mea
    print("NaiveBayes          ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].mea
    print("MLP                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].mea
    print("SVM                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].me
    print()
    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    ### SUBSET ###
    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    df = preprocess(subsetdf)
    df.drop(df.columns[[0]], axis=1, inplace=True)
    # Tokenize the text column to get the new column 'tokenized_text'
```

```python
def word2vec_evaluate(fullsetdf,subsetdf):
    size = 300
    window = 5
    min_count = 1
    workers = 4
    sg = 1
    df = preprocess(fullsetdf)
    df.drop(df.columns[[0]], axis=1, inplace=True)
    # Tokenize the text column to get the new column 'tokenized_text'
    df['tokenized_text'] = [simple_preprocess(line, deacc=True) for line in df['sentence']]
    w2v_model = Word2Vec(df['tokenized_text'].values, min_count = min_count, size = size, workers = workers, window = window, s
    for index, row in df.iterrows():
        model_vector = (np.mean([w2v_model[token] for token in row['tokenized_text']], axis=0)).tolist()
        if index == 0:
            header = ",".join(str(ele) for ele in range(size))
            header = header.split(",")
            word2Vec_df = pd.DataFrame([], columns = header)
          # Check if the line exists else it is vector of zeros
        if type(model_vector) is list:
            line1 = ",".join( [str(vector_element) for vector_element in model_vector] )
    #else:
       # line1 = ",".join([str(0) for i in range(size)])
#line1 = line1.split(',')
    a_series = pd.Series(line1, index = word2Vec_df.columns)
    word2Vec_df =  word2Vec_df.append(a_series,ignore_index=True)
    y_train = pd.DataFrame(fullsetdf['label'])
    X_train = word2Vec_df
    lr_clf = LogisticRegression()
    dt_clf = DecisionTreeClassifier()
    rf_clf = RandomForestClassifier()
    ab_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
    nb_clf = GaussianNB()
    nn_clf = MLPClassifier(random_state=1, max_iter=300)
    svm_clf = svm.SVC(gamma=0.001, C=100.)
    scoring = ['accuracy','precision_macro', 'recall_macro', 'f1_macro']
    print(" *** Full dataset word2vec features *** ")
    scores_lr_clf = cross_validate( lr_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_dt_clf = cross_validate( dt_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_rf_clf = cross_validate( rf_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_ab_clf = cross_validate( ab_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nb_clf = cross_validate( nb_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_nn_clf = cross_validate( nn_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    scores_svm_clf = cross_validate( svm_clf, X_train, y_train, cv=10, scoring=scoring, return_train_score=False)
    print("MODELS              ,Accuracy , Precision  ,  Recall    ,  F-score            ", flush=True)
    print("Logistic Regression ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_lr_clf['test_accuracy'].m
    print("Decision Tree       ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_dt_clf['test_accuracy'].m
    print("Random Forest       ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_rf_clf['test_accuracy'].m
    print("Adaboost            ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_ab_clf['test_accuracy'].m
    print("NaiveBayes          ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_nb_clf['test_accuracy'].m
    print("MLP                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_nn_clf['test_accuracy'].m
    print("SVM                 ,{:.2f}   ,   {:.2f}   ,  {:.2f}    , {:.2f}(+/- {:.2f})".format(scores_svm_clf['test_accuracy'].
    print()
    #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    ### SUBSET ###
```

Results

RQ3: What data transformations do users perform in notebooks to accomplish these tasks?

- We attempted to construct a rough classification of data transformations:
- String operations (e.g. splitting and replacing patterns) were most common, although this is probably heavily dependent on task
- Classifying these transformations was difficult given the variation in code and sheer number of possible transformations, so we went with a broad, incomplete classification
- Overall, data transformations varied heavily

- String operations
    - Removing pattern
    - Replacing pattern
    - Convert to lowercase
    - Convert to uppercase
    - Split
    - Strip
    - Filter strings based on property (length, pattern, etc.)
- Column/array transformations
    - Convert column type
    - Drop column
    - Add new column
    - Apply function to column
    - Replace missing values
    - Select columns
- Row structure
    - Filter rows
    - Add rows
    - Drop rows
    - Select rows
    - Slice rows
- Dataframe level structure
    - Groupby
    - Join tables
    - Get dataframe shape
- Encoding

Discussion

- Big takeaways:
  - Jupyter Notebooks are used for a wide variety of tasks, including some not necessarily related to analyzing one’s data (education and scripting)
  - Notebook users work with a wide variety of data and formats using a wide variety of transformations
  - A significant fraction of notebooks involve comparing between similar approaches in a nonlinear fashion
- Opportunities for future work:
  - How can we better support users of notebooks for **education** and **scripting**?
  - How can we support notebook users in **working nonlinearly** and **comparing between approaches**? **Can we design a notebook not limited to linear 1d execution?**

Thank you!