# June 2, 2018 - Judy

Created a Jupyter notebook that represents all phases of the machine learning process. It leverages many of the best practices demonstrated in class notebooks, applies several common algorithms to our crime data, and systematically records the results in the chart below. The notebook includes:

1. Ingestion, including the http request for current crime data in csv format from: <https://datagate.dc.gov/search/open/crimes?daterange=2years&details=true&format=csv>
2. Data exploration
3. Data wrangling to:
   1. Parse start date into month, day and hour and create three new columns in the dataframe to store the new values
   2. Add a column with a positive value for the longitude
4. Basic data exploration and visualization
5. Deleting features not likely to impact UCR Rank (the target)
6. Saving the wrangled data to a csv file and to a plan text file (without headers and with floats)
7. Loading the text file data and meta.json (features and target names) into a Bunch object
8. Creating a generic fit and evaluation function that takes the data, model name and algorithm parameters as arguments.
9. Evaluates the following models:
   1. KNeighborsClassifier
   2. LogisticRegression
   3. GaussianNB
   4. RandomForestClassifier
   5. SVC

These tests will continue with different parameters, features and targets.

# meta.json

{

"target\_names": {

"1": "violent - homicide",

"2": "violent - sex abuse",

"3": "violent - assault with dangerous weapon",

"4": "violent - robbery",

"5": "property - burglary",

"6": "property - theft other",

"7": "property - theft auto",

"8": "property - motor vehicle theft",

"9": "property - arson"

},

"feature\_names": [

"LATITUDE",

"start\_hour",

"longitude2",

"ucr-rank"

]

}

| **MODEL EVALUATION CHART** | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **MODEL** | **STRENGTHS/ DESCRIPTION** | **WEAKNESSES** | **REGULARIZATION/ PARAMETER NOTES** | **RECS** | **FEATURES** | **TARGET** | **SCORE** |
|  |  |  |  |  |  |  |  |
| BERNOULLINB | Assumes binary data;  How often every feature of each class is not zero;  Sparse count data like text |  | Bigger alpha = less complex; | 68380 | See json | See json |  |
| DECISION TREE |  |  | Tweak tree depth;  See feature importance | 68380 |  |  |  |
| GAUSSIANNB | Apply to continuous data;  Average value and standard deviation of each feature for each class;  Very high dimension data |  |  | 68380 |  |  | 0.284802 |
| KERNELIZED SUPPORT VECTOR MACHINES | Better generalization in low-dimensional spaces |  |  | 68380 |  |  | 0.31721 |
| LINEAR MODELS | good for very large datasets;  good when large number of features compared to number of samples; |  | C – smaller – simpler;  SGDClassifier and SGDRegressor support scalability | 68380 |  |  |  |
| LINEAR REGRESSION | fast to train and predict; work well with small datasets and scale to large |  | Alpha – larger = simpler; | 68380 |  |  |  |
| LINEARSVC |  |  |  | 68380 |  |  |  |
| LOGISTIC REGRESSION |  |  | l1 – only a few features important, so also easier interpretability;  l2 – default;  Use solver=’sag’ for hundreds of thousands or millions of records | 68380 |  |  | 0.25042 |
| MultinomialNB | Assumes count data, e.g. words in sentence; Takes into account average value of each feature for each class; |  | Bigger alpha = less complex;  Sparse count data like text;  Performs better than BernoulliNB with relatively large number of nonzero features (i.e., large documents) | 68380 |  |  |  |
| NAÏVE BAYES CLASSIFIERS | Train faster than linear;  To make a prediction a data point is compared to the statistics for each of the classes, and the best matching class is predicted;  Fast to train and predict;  Very well on hi-dimensional sparse data;  Good on large datasets where training might take too long even with linear model | slightly worse generalization than LogisticRegression and LinearSVC | ; | 68380 |  |  |  |
| RIDGE |  |  | Use solver=’sag’ for hundreds of thousands or millions of records | 68380 |  |  |  |
| KNEIGHBORS |  |  |  |  |  |  | 0.33657 |
| RANDOM FOREST |  |  |  |  |  |  | 0.318458 |