TATE ROSEN &
GRACE MILLS

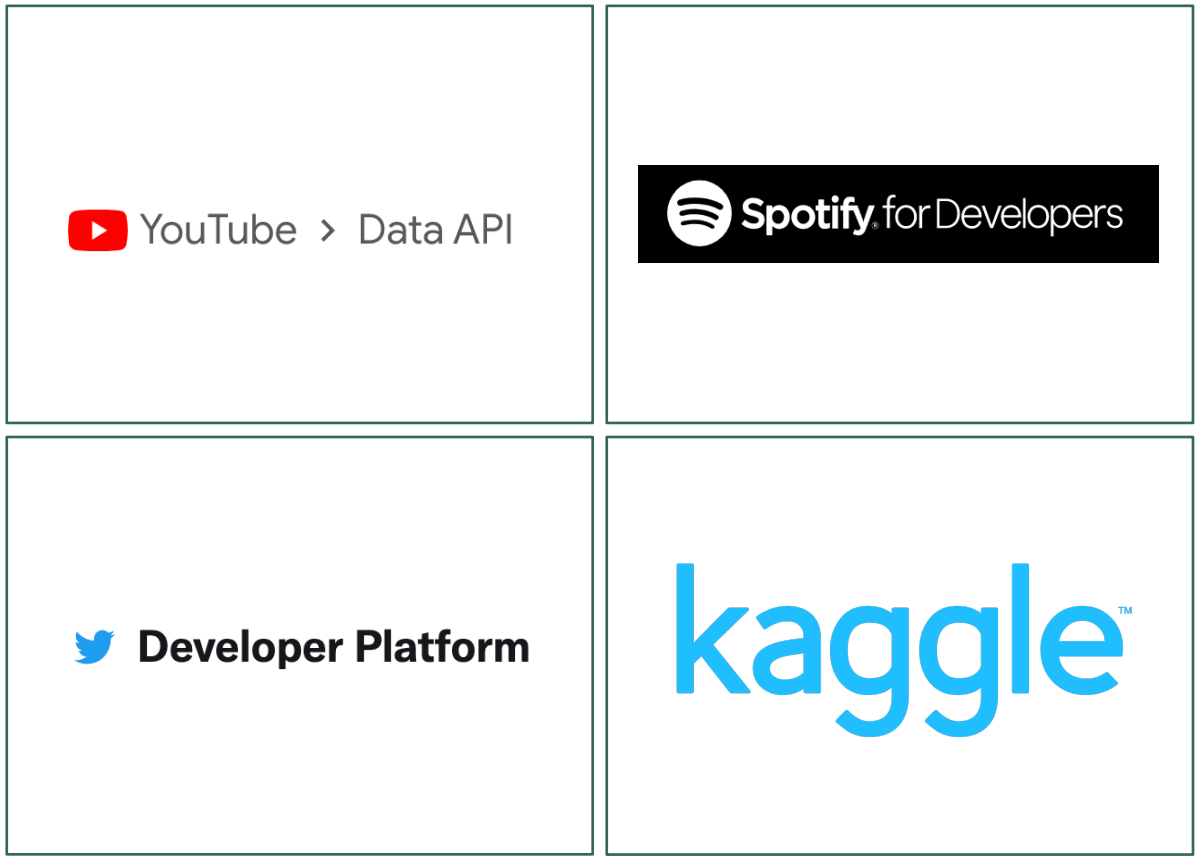
FURMAN
COMPUTER SCIENCE

MACHINE LEARNING ALGORITHMS FOR BIG DATA WITH PYTHON

PROJECT OBJECTIVES

1	Data acquisition	APIs Kaggle.com
2	Data preparation	Missing values, outlier removal, normalization, feature selection, balancing classes
3	Machine learning with Python	Pandas, NumPy, Matplotlib, Scikit-learn Linear regression, clustering, nearest neighbor classification
4	Large scale machine learning with MapReduce	Parallelism with MRJob package

DATASET ACQUISITION



- Application Programming Interfaces (APIs)
  - YouTube, Spotify, Twitter
  - Basic strategy using Python packages
    - Spotipy for Spotify, Tweepy for Twitter
  - Implemented sample code from textbook
    - *A Hands-On Introduction to Data Science*, by Chirag Shah
  - Results:
    - Loaded data from online APIs into files using Python packages (argparse, csv, urllib, apiclient)
    - Learned how to use Python features to execute machine learning algorithms on the acquired datasets
- Data Repositories
  - Kaggle.com
  - UCI Machine Learning Repository
- Target dataset → GPS recorded hikes from hikr.org

THE HIKING DATASET

- Found on Kaggle.com - large dataset about different hikes
- Over 12,000 instances of individual hikes
- Attributes include length, elevation, speed, uphill/downhill, username, bounds, hike name, difficulty, etc.
- Possible machine learning questions:
  - Predict the level of difficulty of a new hike?
  - Determine which hikers might be most compatible?

DATA PREPARATION, PART 1

Difficulties in opening file to read/edit: the hiking data had a .csv extension, but loaded incorrectly due to attributes with XML values → we couldn't see the rows of data

Solution: used Pandas to delete the column whose entries were causing the issue, which led to the data being visible in the typical .csv format

Difficulties in loading data: NumPy could not load data correctly due to the problematic column, and the first row had column names that were strings instead of floats

Solution: used Pandas instead of NumPy to load the data and specify the desired columns, and created a copy of the data frame as a new file with no index names to use only floats

MACHINE LEARNING WITH PYTHON

- Rich library of packages for data manipulation and machine learning
  - Pandas

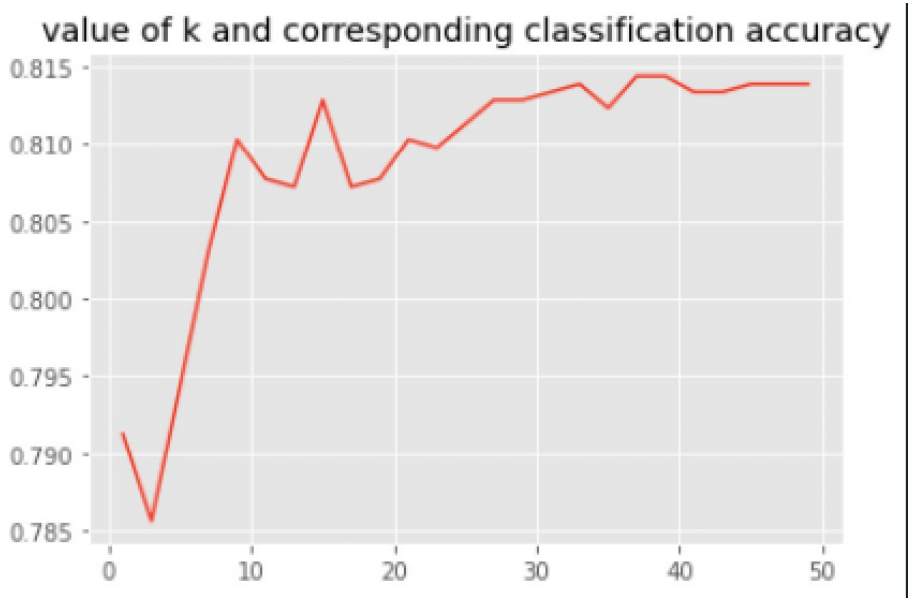
```
# Create dataframe
df = pd.read_csv("hiking.csv")
```

    - Gives user the ability to load a dataset into a dataframe and manipulate it
    - Experienced difficulties with loading data using NumPy's loadTxt, solved with Pandas
    - Ex: kNN algorithms on multiple datasets, clustering adjustments, and dropping NaN values
  - NumPy
    - Allows the creation of NumPy arrays and includes many mathematical functions to manipulate them
    - Originally used in the textbook's clustering and kNN examples, but gave many issues with hiking dataset

```
# count how many instances were correctly classified
correct = np.where(prediction == y_test, 1, 0).sum()
```

MACHINE LEARNING WITH PYTHON CONT.

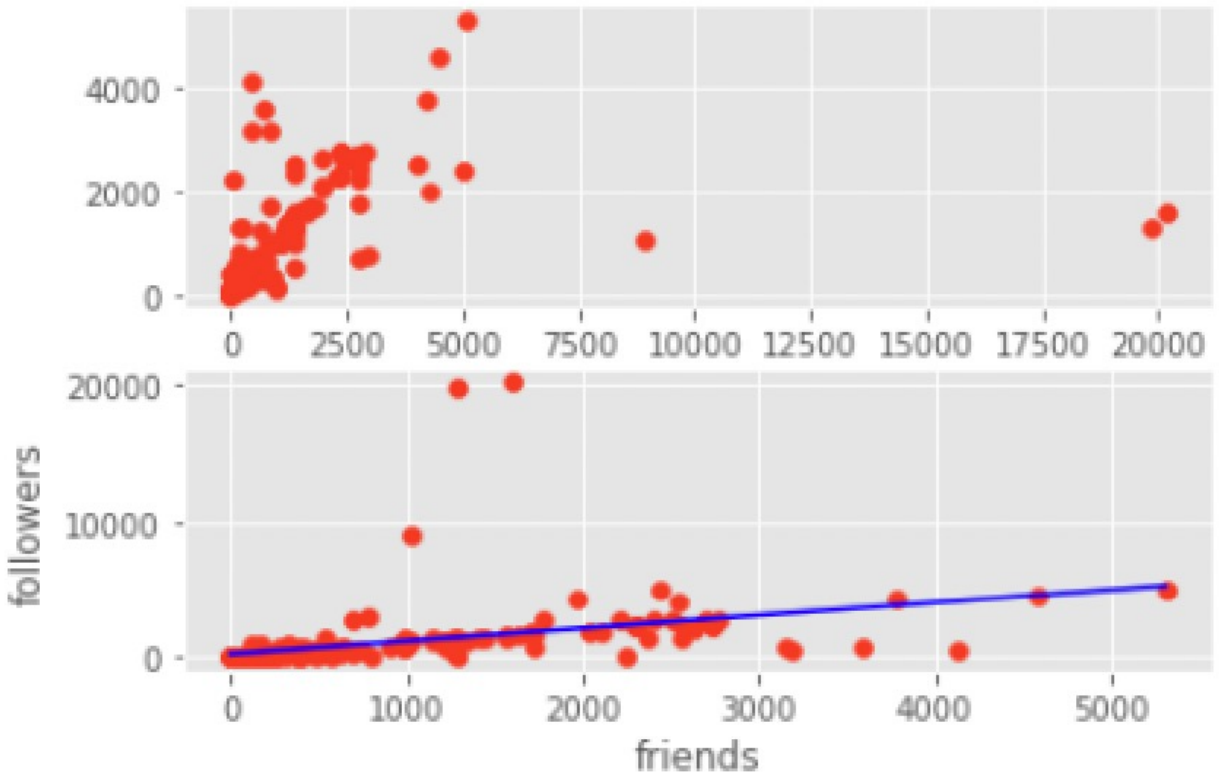
- Rich library of packages for data manipulation and machine learning
  - Matplotlib
    - Allows the creation of different types of plots and provides functions to alter their appearance and contents
    - Plot for kNN with wine dataset
  - Scikit-learn
    - Machine learning library for Python
    - Includes various classification, clustering and regression algorithms
    - kNN, Support Vector Machines (SVM), k-Means



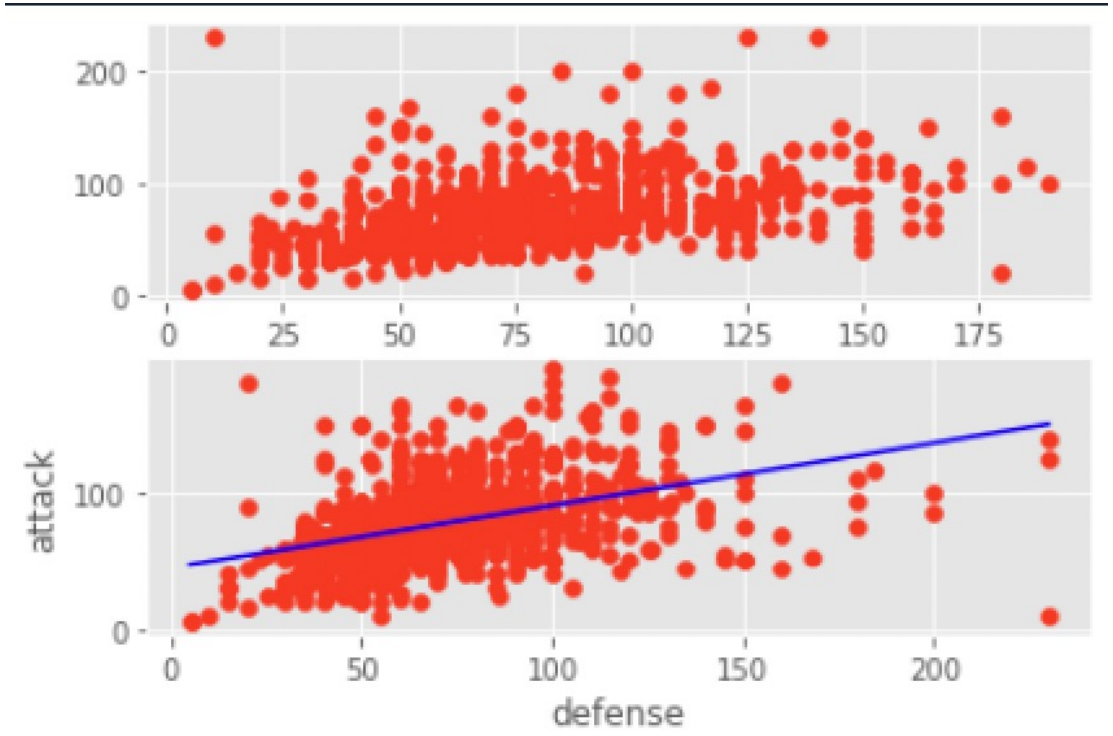

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

```
classifier = KNeighborsClassifier(n_neighbors = 3)
classifier.fit(X_train, y_train)
```

LINEAR REGRESSION: STATSMODELS.API PACKAGE

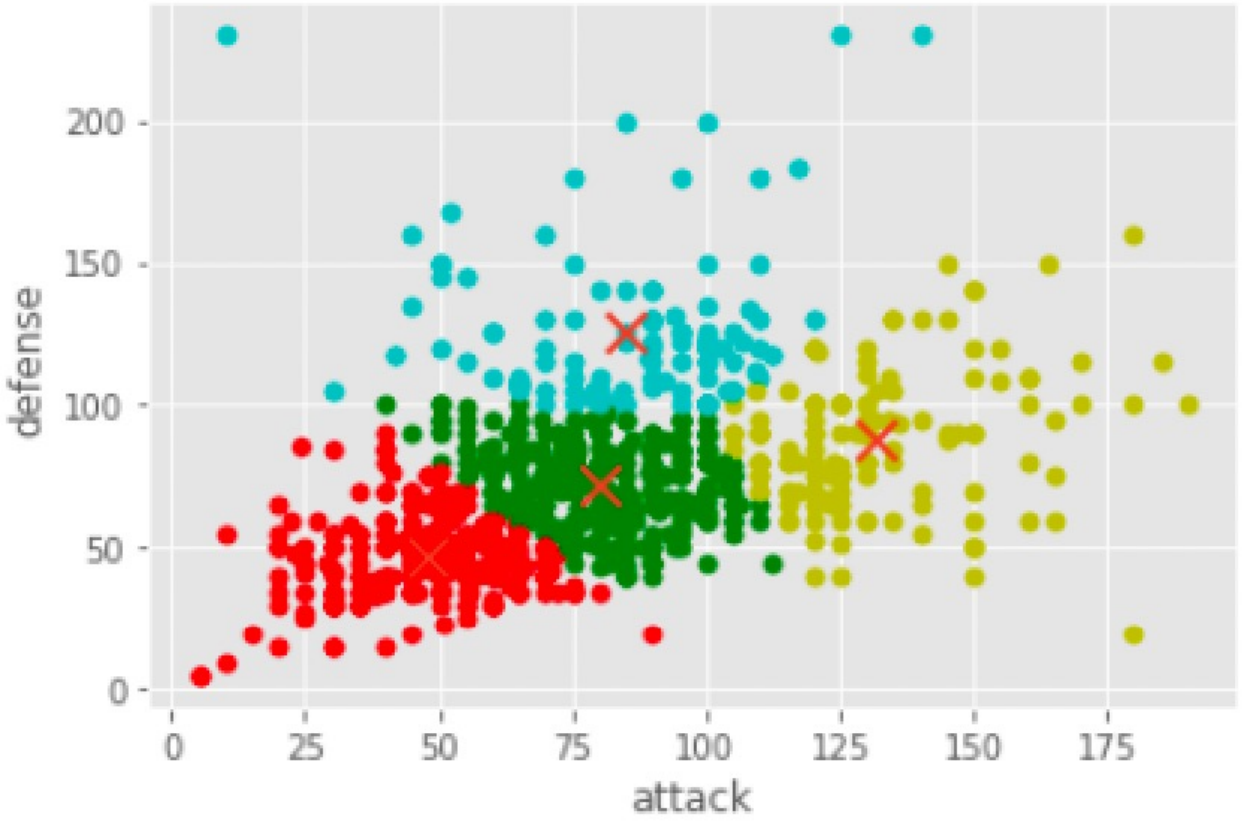


Twitter

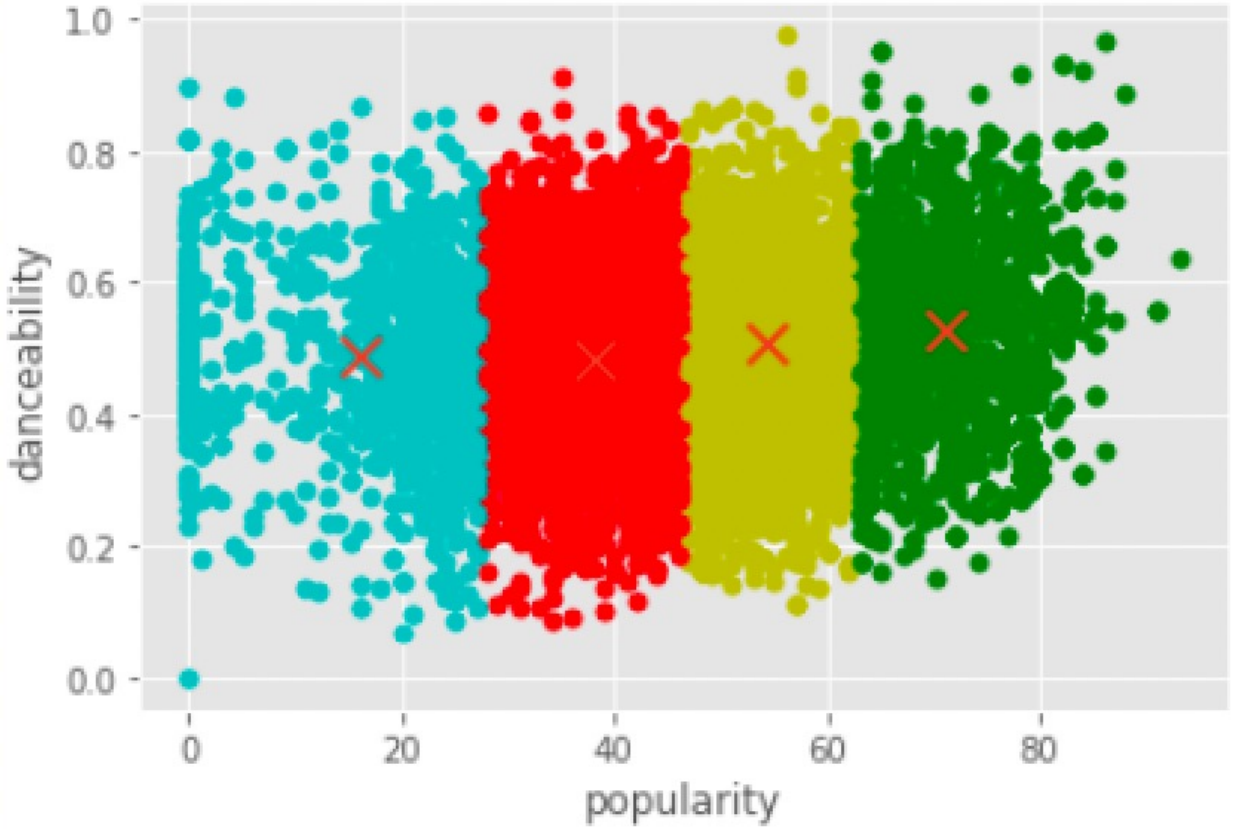


Pokemon

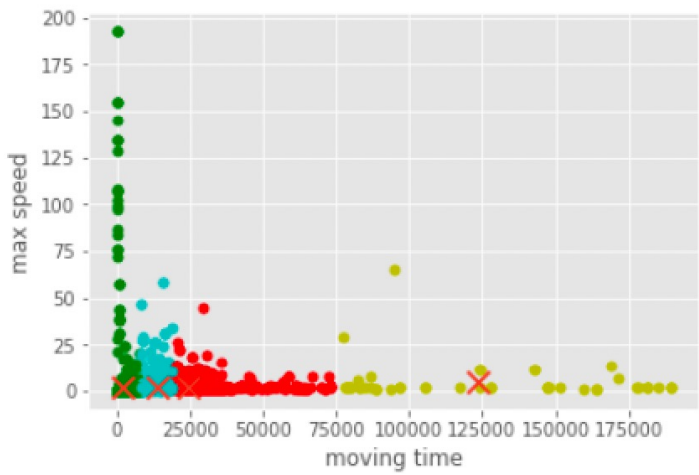
CLUSTERING: KMEANS FROM SKLEARN.CLUSTER PACKAGE

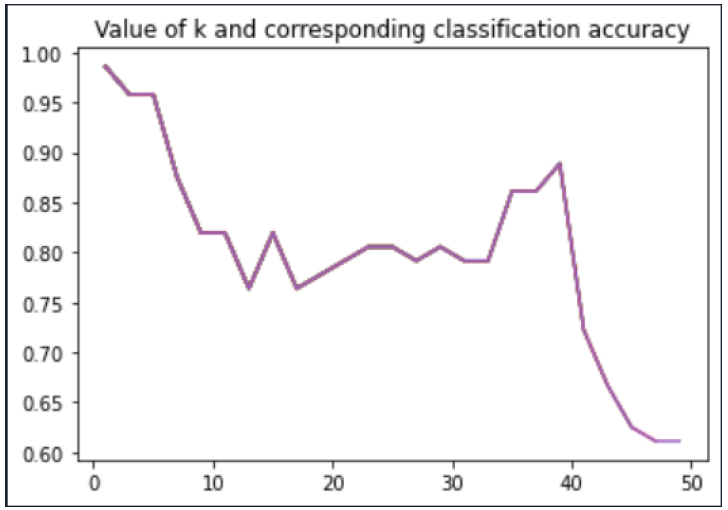


Pokemon



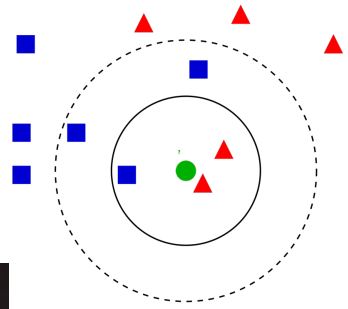
Spotify





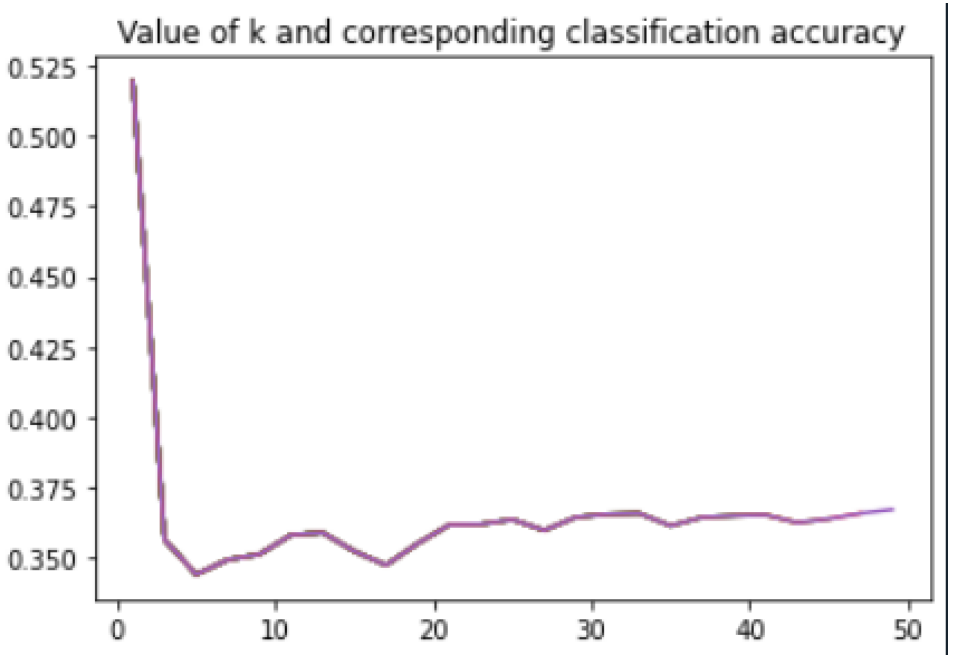
MINING THE HIKING DATASET

- Clustering
  - Original code included NumPy and was not able to work with the hiking dataset
  - We altered the code to use Pandas instead, which gave results
- Nearest neighbor classification
  - Uses instances closest to current one in order to classify the current one into a certain category
    - Euclidean distance
    - Neighborhood size (value of k): the number of neighbors to include in the majority voting process for a new instance
  - Stars dataset
    - Attributes like Temperature, Color, Spectral Class
    - Almost 100% accuracy k=1,dropped, spiked again at k=40



MINING THE HIKING DATASET WITH KNN

- Question: Can the difficulty of a new hike be predicted using nearest neighbor classification?
- Distance metric
  - Minkowski: option of either Euclidean or Manhattan
- Original results (k=1)
  - classifier = KNeighborsClassifier(n_neighbors = 1, p=1, metric='minkowski')
    classifier.fit(X_train, y_train)
  - Accuracy measure
    - 30% holdout method (70% of dataset for training; 30% for testing)
    - Test data selected randomly using train_test_split()
- method from sklearn package
- Around 50% accuracy in predicting the difficulty level of a hike in the test dataset
- ***Can we do better?***



DATA PREPARATION, PART 2

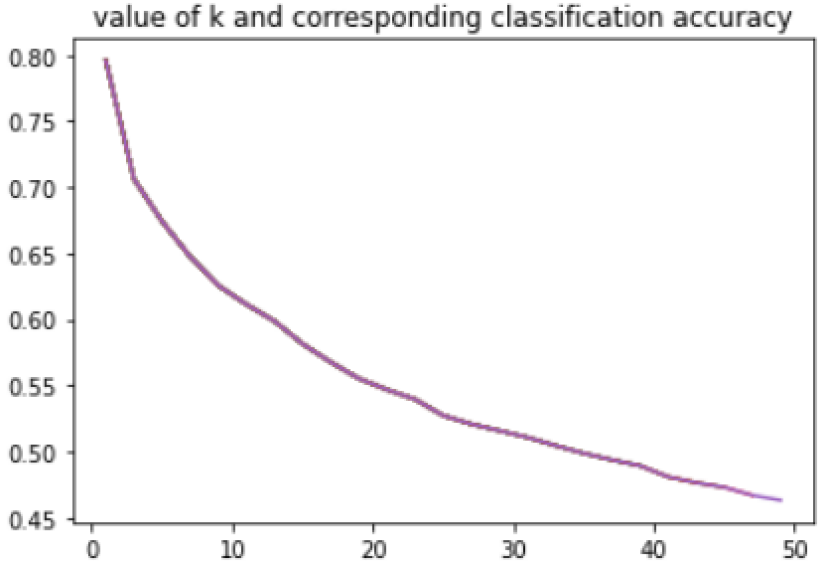
- ~50% accuracy is acceptable with 13 difficulty levels, but not optimal
- Outlier removal
  - Calculate avg_speed (length_3d/moving_time) and remove instances with avg_speed > 2.5 meters/sec, which are either inaccurate or not hiking activities
  - Remove instances with suspiciously low elevation measures
- Feature selection
  - Convert difficulty attribute to numeric value
  - Use df.corr() method to measure correlations between input attributes and difficulty
  - Select max_elevation, min_elevation, uphill, downhill as potential most predictive

	difficulty_num
length_3d	0.035156
max_elevation	0.460304
uphill	0.230520
moving_time	0.095606
max_speed	-0.004895
min_elevation	0.238040
downhill	0.183401
length_2d	0.035156
avg_speed	-0.152020
difficulty_num	1.000000

DATA PREPARATION, PART 2 CONT.

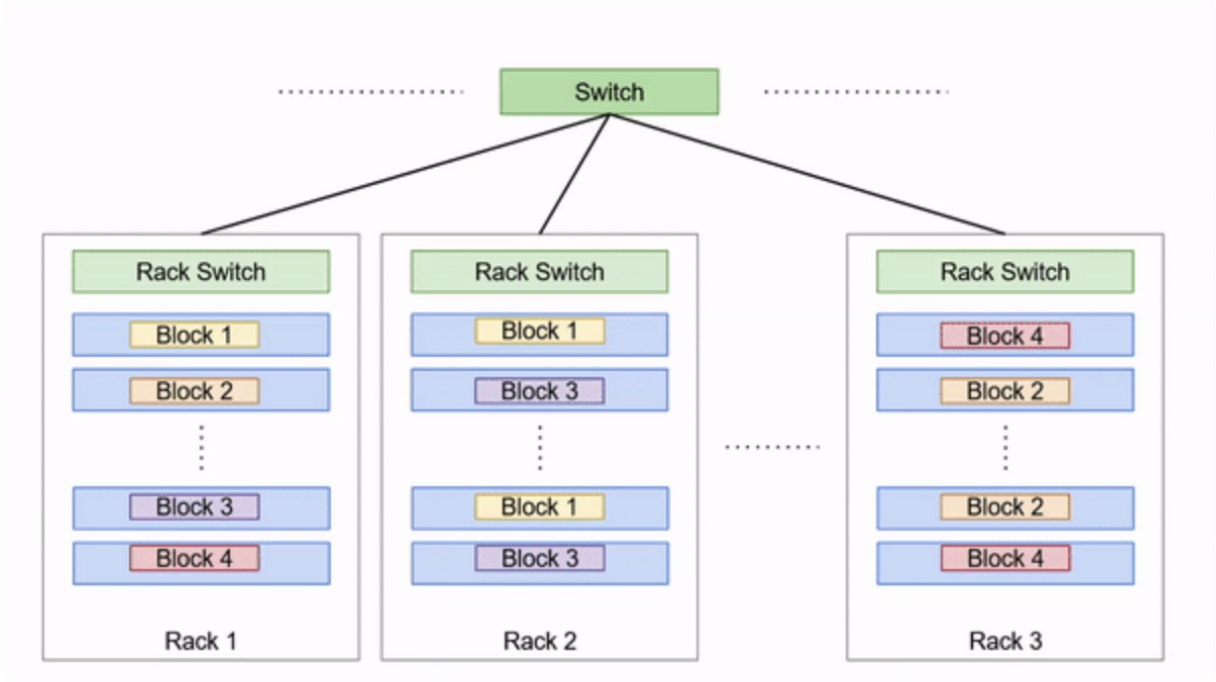
- Normalizing attribute values
  - Selected input attributes have different scales
    - same absolute difference in values may be weighted incorrectly
  - normalize() method from the preprocessing package converts all to [0,1] scale
- Balancing class values
  - Count instances of class values for distribution
  - 30.17% are "T2" and 24.87% are "T3". 44.96% of dataset for other 11 difficulty levels
  - Nearest neighbor prone to overpredict T2 and T3
  - SMOTE() method from imblearn package *oversamples* so that class values are equal
- Results
  - Cumulative effect of preprocessing results in accuracy improvement to 77.70% for k=1

```
Class=1, n=2785 (7.692%)
Class=2, n=2785 (7.692%)
Class=0, n=2785 (7.692%)
Class=3, n=2785 (7.692%)
Class=6, n=2785 (7.692%)
Class=4, n=2785 (7.692%)
Class=9, n=2785 (7.692%)
Class=8, n=2785 (7.692%)
Class=5, n=2785 (7.692%)
Class=7, n=2785 (7.692%)
Class=10, n=2785 (7.692%)
Class=12, n=2785 (7.692%)
Class=11, n=2785 (7.692%)
```

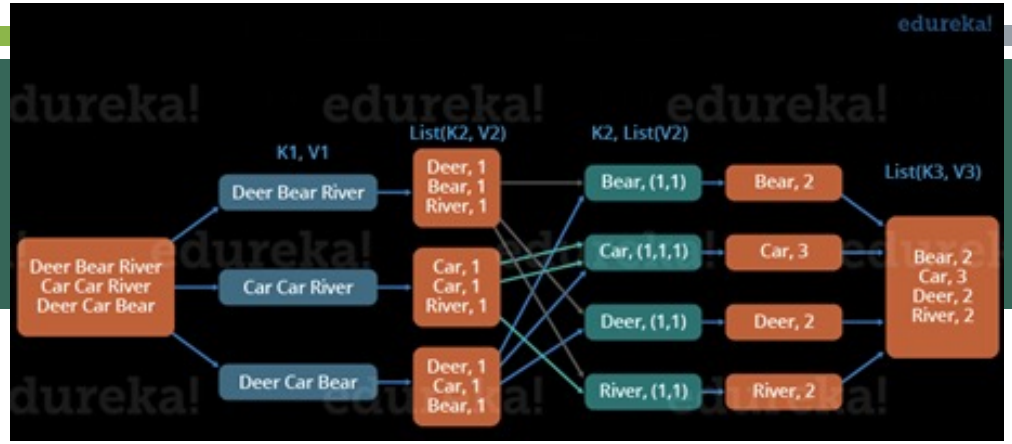


LARGE SCALE MACHINE LEARNING WITH MAPREDUCE

- Benefits of parallelism with large datasets
- Programming paradigm
  - **Hadoop Distributed File System (HDFS)**
    - designed to store large amounts of data
    - data is divided into chunks and stored across multiple racks
  - **MapReduce**
    - programming framework that performs parallel processing on large datasets
- AWS (Amazon Web Services)
  - Among many other features, allows a MapReduce program to run on multiple computers to provide results in less time and/or for larger amounts of data
  - Applicability to this problem (i.e., can spin up a Hadoop cluster and run code on it)



MAPREDUCE PARADIGM



- **Simple MapReduce Jobs (Counters):**

```python
from mrjob.job import MRJob

class hikeusercounter(MRJob):

    def mapper(self, _, line):
        (_id, length_3d, user, max_elevation, uphill, moving_time, max_speed,
         difficulty, min_elevation, downhill, name, lenght_2d) = line.split(',')

        yield user, 1


    def reducer(self, user, occurrences):
        yield user, sum(occurrences)
```

mapper: count each user once
reducer: sum up how many times each user appeared
output: user's name, times appeared

```python
from mrjob.job import MRJob

class hikenamecounter(MRJob):

    def mapper(self, _, line):
        (_id, length_3d, user, max_elevation, uphill, moving_time, max_speed,
         difficulty, min_elevation, downhill, name, lenght_2d) = line.split(',')

        yield name, 1


    def reducer(self, name, occurrences):
        yield name, sum(occurrences)
```

mapper: count each hike name once
reducer: sum up how many times each hike name appeared
output: hike name, times appeared

MAPREDUCE CODING

Word Count:

```
"communication" 5
"community"     12
"commute"       14
"commuted"      1
"commuting"     6
"companies"     31
"companions"    1
"company"       115
```

Simple hiking examples (counting instances of each difficulty, users, etc.):

```
"Pizzo dell'Uomo"    3
"Pizzo dell\u2019Uomo (2663 m)"   1
"Pizzo dell\u2019Uomo \u2013 anticima N (2585 m)"  1
"Pizzo della Croce o Monte Crocione (1491 m) dalla Valle di Muggio" 2
"Pizzo della Moriana (2631 m) - Valsesia"  1
"Pizzo della Pieve e Grignone"    2
"Pizzo della Presolana (2521m)"   1
"Pizzo della Sella bell' anello in Val Cairasca"   1
"Pizzo delle Pecore e tentativo a Cima Saler (Val Grande)" 1
"Pizzo delle Pecore m. 2018 Bassa Ossola"  1
```

```
"johnny68"   6
"jottlieb"   1
"justus" 26
"kardirk"    28
"kleopatra"  30
"kneewoman"  5
"kopfsalat"  72
"lampbarone" 2
"laponia41"  57
"larsschw"   1
"leuti"  4
"lila"   28
```

Movie Recommender System:

```
"Parent Trap, The (1961)"       ["Little Women (1994)",0.9627420015567533,34]
"Parent Trap, The (1961)"       ["Sabrina (1954)",0.9628100516947542,15]
"Parent Trap, The (1961)"       ["Dead Poets Society (1989)",0.9628751629416521,58]
"Parent Trap, The (1961)"       ["101 Dalmatians (1996)",0.9629801022299663,27]
"Parent Trap, The (1961)"       ["Return of the Pink Panther, The
(1974)",0.9630163969504121,26]
"Parent Trap, The (1961)"       ["Arsenic and Old Lace (1944)",0.9630196215066396,32]
"Parent Trap, The (1961)"       ["Hunchback of Notre Dame, The
(1996)",0.9631034602166914,22]
```

MINING THE HIKING DATASET IN PARALLEL FOR ASSOCIATION RULES

- We wanted to create a hiking association system similar to the MR movie recommender system
- Extract usernames and several hiking attributes to determine similarity

Mappers:

Mapper 1: extract attribute values and create key/value pairs: key = difficulty, value = (user, [list of attributes])
Mapper 2: create user pairs within each difficulty rating
Mapper 3: sort results by username

Reducers:

Reducer 1: append user/attribute lists by difficulty
Reducer 2: compute similarity score
Distance metric: Cosine similarity
Reducer 3: output the results

```
def steps(self):
    return [
        MRStep(mapper=self.mapper_parse_input,
               reducer=self.reducer_attributes_by_diff),
        MRStep(mapper=self.mapper_create_user_pairs,
               reducer=self.reducer_compute_similarity),
        MRStep(mapper=self.mapper_sort_similarities,
               reducer=self.reducer_output_similarities)]
```

MINING THE HIKING DATASET IN PARALLEL FOR ASSOCIATION RULES CONT.

```
for lengthX, lengthY in lengthPairs:
    sum_xx += np.dot(lengthX, lengthX)
    sum_yy += np.dot(lengthY, lengthY)
    sum_xy += np.dot(lengthX, lengthY)
```

```
"olethros"      ["cristina",0.9787364341907547,1074]
"pizzo1954"     ["cristina",0.9744371501881771,780]
"cristina"      ["olethros",0.9787364341907547,1074]
"cristina"      ["Abominevole",0.9822285661501574,512]
"Marchino"      ["cristina",0.9716312526748905,768]
```

- Discussion
  - Ideal level of support determined to be around 300 pairs
  - Interpretation of results: **Which users would be good hiking partners based on reported hikes?**
  - Length-only version uses length values to determine similarity
  - Another version uses five different attributes (length, uphill, downhill, max speed, and moving time) to determine similarity
  - Our hypothesis is that the 5-attribute version gives more accurate results because it takes more aspects of each hike into account when calculating compatibility

CONCLUSIONS

Issues of acquiring and cleaning datasets:

- Limited amount of data from APIs at one time. Ex: Only 180 tweets from Twitter at a time.
- Many have missing values or an unbalanced dataset and need adjustments
- Kaggle.com is a good resource to find interesting and ready-to-go datasets

Lessons from Python data mining libraries:

- Has many libraries/packages for any problem you are looking to solve
- If one library doesn't work for your data, another one will

Suitability of arbitrary datasets for data mining

- Not every dataset is fit for (good) data mining.
- If one algorithm has a low accuracy, there's likely a better one.
- In the hiking dataset, the correlation between attributes was very low, so the results won't yield a high accuracy.

Parallelism through MapReduce:

- Challenges: Not every type of problem can be solved with MapReduce.
- Promise of significant benefits: Faster computing only with *extremely* large datasets
- Cheap to use AWS for cluster computing

FUTURE WORK

Application of other machine learning algorithms to hiking dataset

- Not every algorithm is going to yield high accuracies for every dataset, even if you *improve* that dataset
- Others to try: kNN Regression, Naīve Bayes, Decision Trees, etc.

Application of MapReduce to larger datasets

- Running our hiking partner recommendation program with AWS
- Faster with 4+ nodes rather than locally.

Using other parallel processing frameworks

- Spark: an open-source unified analytics engine for large-scale data processing

THANK YOU!

WE HOPE YOU ENJOYED THE PRESENTATION AND WOULD BE HAPPY TO ANSWER ANY QUESTIONS.