Data Foundations: Vectors and Machine Learning

Instructor: Anthony Rios

Outline

Data Objects and Attribute Types

A **data object** represents an entity. Examples include...

- Customers
- Students/Professors/Courses
- Tweets
- Images
- Genes, Drugs, Procedures

A **attribute** is a data field (synonyms dimension/feature/variable)

Types of Attributes

`https://www.youtube.com/watch?v=N9fDIAflCMY`

- **Nominal** - hair_color (black, blonde, red, green?), martial_status (single, married, divorced, widowed,...), occupation (teacher, dentist, programmer,...)

- **Binary (Boolean)** - smoker (yes or no), medical tests (positive or negative)

- **Ordinal** - drink_size (short, tall, grande, venti), grade (A, B, C, D, F), professional_rank (assistant, associate, full)

- **Numeric** - temperature (70°F), speed (400 mph)

Nominals: How should we represent a nominal?

As a number?

- **As integers?** e.g., hair_color (black $= 0$, blonde $= 1$, red $= 2$)
  - Not the best choice because there is no inherent order to the categories?
- **As a vector?**
  e.g., hair_color (black $= [1, 0, 0]$, blonde $= [0, 1, 0]$, red $= [0, 0, 1]$
  - This is referred to as one-hot encoding.

Nominals in Python: Method 1

```
>>> import numpy as np
>>> vec = np.zeros((3,)) # Assume 4 colors: Blue, Red, Green
>>> classes = ["blue", "red", "green"]
>>> vec[classes.index("green")] = 1
>>> vec
,1
```

For multiple **objects** (examples), the vectors above can be generated for each, appended to a list, then cast to a numpy array.

Nominals in Python: Method 2

>>> **from sklearn.feature_extraction import DictVectorizer**

>>> datasetDicts = [{'age':1, 'hair_color':'blue'},
{'age':2, 'hair_color':'green'}]

>>> vec = DictVectorizer(sparse=False) **# In general it is better to use sparse=True**

>>> dataset = vec.**fit_transform**(datasetDicts) **Takes a list of dicts**

>>> dataset **# dataset is a numpy array with shape (2, 3)**
array([[1., **1., 0.**],
[2., 0., 1.]])

>>> vec.feature_names_
['age', 'hair_color=blue', 'hair_color=green']

Nominals in Python: Method 2

How do we vectorize new data? Use the .transform() method without fit_

```
>>> new_data = [{'age':3, 'hair_color':'purple'}]

>>> new_X = vec.transform(new_data)

>>> new_X
[3., 0., 0.]])
```

Nominals in Python: Method 2

DictVectorizer handles **nominal**, **binary**, and **numeric features**

```
>>> data = [{'age':3, 'hair_color':'purple', 'is_smoker':0}]
```

Store **nominals** as a string, **binary** variables as a 0 (absent) or 1 (present), and **numeric** features as an integer/float.

What about multiple attributes?

What if someone has **multiple** hair colors?

What about multiple attributes? Method 1

What if someone has **multiple** hair colors?

```
>>> vec = np.zeros((3,)) # 3 hair colors red, green, and blue

>>> classes = [”blue“, ”red“, ”green“]

>>> vec[classes.index(‘red’)] = 1

>>> vec[classes.index(‘green’)] = 1

>>> vec[classes.index(‘blue’)] = 1
```

The person has red, green AND blue hair.

What about multiple attributes? Method 2

What if someone has **multiple** hair colors?

```
>>> from sklearn.preprocessing import MultiLabelBinarizer
```

```
>>> mlb = MultiLabelBinarizer()
```

MultiLabelBinarizer takes a list of sets as input to the fit transform method.

```
>>> mlb.fit_transform([set(['red', 'green']), set(['blue'])])
array([[0., 1., 1.],
       [1., 0., 0.]])
```

How can we combine multiple sets of attributes?

Assume two sets of attributes in the form of 2 matrices:

```
>>> atts1
array([[0., 1., 1.],
       [1., 0., 0.]])

>>> atts2
array([[1., 1., 0.],
       [2., 0., 1.]])

>>> combine_atts = np.hstack([atts1, atts2]) # Horizontal stack

>>> combine_atts
array([[0., 1., 1., 1., 1., 0.],
       [1., 0., 0., 2., 0., 1.]])
```

Loading Data from a CSV: All Numeric data

myData.csv

```
label,feature 1,feature 2
positive,0.1,1
positive,1,7
positive,3,4
```

example.py

```
import csv
import numpy as np
X = [] # Will be a list of lists
y = [] # will be a list
with open('myData.csv') as inFile:
        iCSV = csv.reader(inFile, delimiter=',')
        next(iCSV)
        for row in iCSV:
                X.append([float(x) for x in row[1:]]) # get features
                y.append(row[0]) # get class
X = np.array(X) # convert to numpy array for scikit-learn
y = np.array(y) # convert to numpy array for scikit-learn
```

Exercise 1

Write code to load the data in the "iris.csv" into Numpy arrays.

The first 4 columns are the features/attributes. The last column is the class. Simply load the class as a list of strings. Don't forget to convert the dataset into a numpy array. You can use either DictVectorizer or the CSV method on the previous slide to load the features.

Machine Learning

https://www.youtube.com/watch?v=1Hx8_BAfgj8

Basic machine learning problems

- Classification and Regression
  - Linear Methods (Lasso, Ridge, Linear SVM, Logistic Regression)
  - Non-linear Methods (Neural Networks, kernel methods, kNN, ..)
- Clustering (Unsupervised Learning)
  - k-Means
  - Hierarchical Clustering

Classification

Widely used task: given data determine the class it belongs to. The class will lead to a decision or outcome.

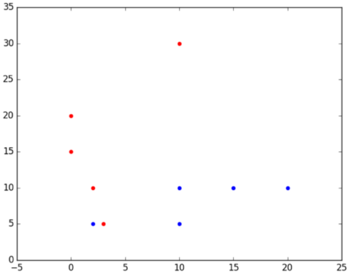
Used in many different places:

- DNA and protein sequence classification
- Insurance
- Weather
- Experimental physics: Higgs Boson determination

Data – Vectors

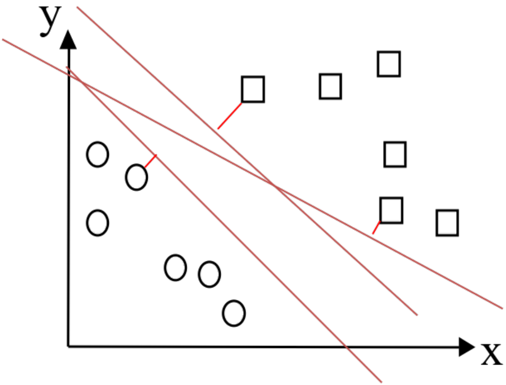
We think of data as vectors in a fixed dimensional space. The entire dataset forms a matrix.

Heart disease	Cigarettes per day	Exercise per day (mins)
1	10	10
1	2	5
1	20	10
1	10	5
1	15	10
0	10	30
0	2	10
0	3	5
0	0	20
0	0	15



Linear Classification Models

One inear model is an SVM. SVMs learn a **hyperplane** in vector space
The margin is the **minimum** distance of **all closest point** (misclassified have negative distance)
The support vector machine results in the hyperplane with **largest margin**



Logistic Regression: Predict House Price

Goal: Predict the whether (yes or no) a home will sell within 6 months

What information is useful to predict a home's price?

- Size of the home (Square Feet; Number)
- Size of the Lot (Square Feet; Number)
- Number of Bedrooms (Number)
- Number of Bathrooms (Number)
- City the home is located (Categorical)
- Avg. price of homes in the same neighborhood

Logistic Regression

$$P(y = 1|x, \beta) = \frac{1}{1 + \exp(-\sum_{i=1}^{F} x_i \beta_i)}$$

Output Space

$$y \in [0, 1]$$

Logistic Regression

$$P(y = 1|x, \beta) = \frac{1}{1 + \exp(-\sum_{i=1}^{F} x_i\beta_i)}$$

Output Space

$$y \in [0, 1]$$

$$\sum_{i=1}^{F} x_i\beta_i = x_1\beta_1 + x_2\beta_2 + \ldots + x_F\beta_F$$

Goal: Learn the parameters β_i using labeles data!

Logistic Regression: Features/coefficients

x = **feature vector** β = **coefficients**

Feature	Value
Home Size	2500
Lot Size	18000
# Bedrooms	4
# Bathrooms	2.5
Loc-San Antonio	1
Loc-New York City	0
Loc-Austin	0
Avg. Price	100000
BIAS	1

Feature	β
Home Size	0.01
Lot Size	0.03
# Bedrooms	1.4
# Bathrooms	3.1
Loc-San Antonio	1.2
Loc-New York City	0.5
Loc-Austin	-3.0
Avg. Price	-0.8
BIAS	-0.1

Logistic Regression: Features

	BIAS	Home Size	Lot Size
β	-0.1	3.1	1.2

	BIAS	Home Size	Lot Size	$a = \sum x_i \beta_i$	$\exp(-a)$	$1/(1+\exp(-a))$	true y
x^1	1	1	1	3	0.05	95.2%	1
x^2	1	1	1	4.2	0.015	98.5%	1
x^3	1	0	0	-0.1	1.11	47.4%	0

Loading Data with Scikit-Learn

>>> **from sklearn import datasets # Used to load toy datasets in sklearn**

>>> raw_data = datasets.load_wine() **# Loads a dictionary with data**

>>> X = raw_data["data"] **# A matrix of data**

>>> X.shape **Tuple with numb. of rows and columns (features) in X**
(**178**, **13**)

>>> y = raw_data["target"] **# The classes**

>>> y.shape **# Vector with class index for each row of X**
(**178**,)

Loading Data with Scikit-Learn

```
>>> X[0]
array([1.423e+01, 1.710e+00, 2.430e+00, 1.560e+01, 1.270e+02,
2.800e+00, 3.060e+00, 2.800e-01, 2.290e+00, 5.640e+00, 1.040e+00,
3.920e+00, 1.065e+03])

>>> y
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Training a Model

The following snippets assume we have loaded the data and have Train/Test splits.

```
>>> from sklearn.svm import SVC
>>> clf = SVC() # init. default SVM model
>>> clf.fit(X_train, y_train) # contrats! You trained an SVM!!!
```

It is important to fiddle with machine learning algorithm parameters to understand how they effect performance.

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Making Predictions and Evaluation

To make predicts on held-out data you can do the following:

```
>>> y_pred = clf.predict(X_test)

>>> from sklearn.metrics import accuracy_score

>>> accuracy = accuracy_score(y_test, y_pred)

>>> print("accuracy: {}".format(accuracy))
0.89
```

https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics

What you need to know

- Why you can't use "training-set-error" to estimate the quality of your learning algorithm on your data.
- Why you can't use "training-set-error" to choose the learning algorithm
- Leave-one-out cross-validation
- k-fold cross-validation
- what is grid-search?
- Basic idea behind the bias variance tradeoff

Two Goals

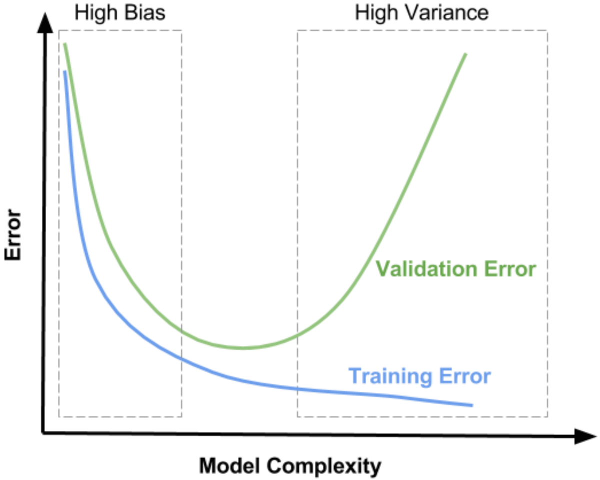
Model Selection: estimate the performance of different models in order to choose the best one.

Model Assessment: having chosen a final model, estimate its prediction error on new data.

Model Selection: Training Data Only

```
>>> from sklearn.svm import SVC

>>> clf = SVC()

>>> clf.fit(X_train, y_train)

>>> y_pred = clf.predict(X_train)

>>> from sklearn.metrics import accuracy_score

>>> accuracy = accuracy_score(y_train, y_pred) # Should we expect this accuracy on new data?
0.99
```

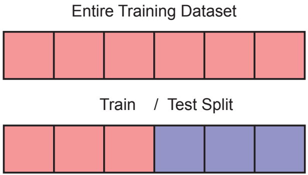
Bias Variance Trade Off



The Test(Validation)-Only Split

example.py

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
max = -1 C_range = [0.001, 0.01, 0.1, 1., 10.] # SVM regularization params
# split data into training and test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
for x in C_range:
        clf = SVC(C=x)
        clf.fit(X_train, y_train)
        preds = clf.predict(X_test)
        acc = accuracy_score(y_test, preds)
        if acc > max:
                print("New best {}".format(acc))
                max = acc
```



The Test(Validation)-Only Split

- In problems where we have a sparse dataset we may not be able to afford the "luxury" of setting aside a portion of the dataset for testing

- Since it is a single train-and-test experiment, the holdout estimate of error rate will be misleading if we happen to get an "unfortunate" split

Exercise 2

Using the iris data you loaded in Exercise 1, do the following:

- Use train_test_split() to split the iris dataset. (use 0.2 for the test_size)
- train an SVM on the train split and evaluate using accuracy on the test split.
- Fiddle with the parameters of the SVM to see how it effects the performance.
- Calculate the accuracy on the train split.

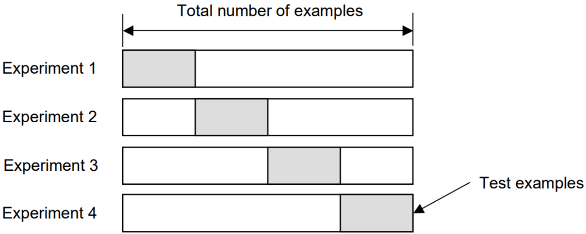
Next, try using a different classifier, a random forest, and see how it compares to the SVM

`https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

Note that this is a toy dataset, so all scores will be high.

K-Fold Cross-Validation

- Create K-fold partition of the dataset
  - For each of the K experiments, use K-1 folds for training and the remaining one for testing.



- The final error (evaluation measure) can be estimated as

$$E = \frac{1}{K} \sum_{i=1}^{K} E_i$$

K-Fold Cross-Validation with Scikit-Learn

>>> **from sklearn.model_selection import cross_val_score**

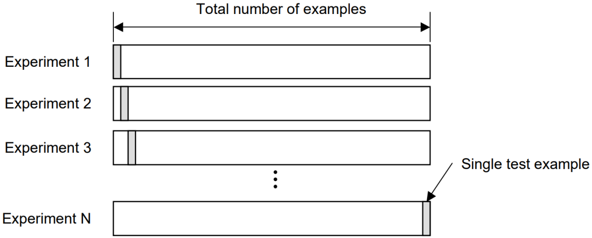
>>> from sklearn.linear_model import Lasso **# Regression Model**

>>> r = Lasso()

>>> print(**cross_val_score**(r, X, y, scoring='neg_mean_squared_error', cv=3))
[0.33150734 0.08022311 0.03531764]

Leave One Out Cross-Validation

- For a dataset with N examples, perform N experiements
  - For each experiment use N-1 examples for trianing and the remaining examples for testing



- The final error (evaluation measure) can be estimated as

$$E = \frac{1}{N} \sum_{i=1}^{N} E_i$$

Leave One Out Cross-Validation

```
>>> from sklearn.model_selection import cross_val_score
```

```
>>> from sklearn.linear_model import Lasso # Regression Model
```

```
>>> r = Lasso()
```

```
>>> print(cross_val_score(r, X, y, scoring='neg_mean_squared_error',
cv=X.shape[0]))
[0.33150734 0.08022311 0.03531764]
```

How many folds are needed?

- With a large number of folds
  - + The bias of true error rate is small
  - - Variance of true error rate will be large
  - - Large computational time (many experiments)
- With a small number of folds
  - + Computation time reduced
  - + Variance of estimator will be small
  - - The bias of estimator will be large (Conservative or higher than true error rate)
- In practice, the choice of the number of folds depends on the size of the dataset
  - For large datasets, 3-fold CV will be quite accurate
  - For very small datasets, we may have to use leave-one-out in order to train on as many examples as possible
- A common choice for K-Fold CV is K=10

Grid-Search in Scikit-Learn

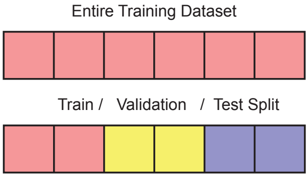
```
>>> from sklearn.model_selection import GridSearchCV
>>> from sklearn.linear_model import SVC
>>> parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
>>> c = SVC()
>>> clf = GridSearchCV(c, parameters, scoring='f1_micro', cv=3)
```

Grid-Search in Scikit-Learn

```
>>> clf = GridSearchCV(r, parameters, scoring='f1_micro', cv=3)

>>> clf.fit(X, y)

>>> print(clf.best_params_)
{'kernel':'linear', 'C':1}

>>> print(clf.best_score_) # Mean of CV scores
0.42
```

Model Assessment

- Three-way data splits
  - **Training set**: A set of examples for learning
  - **Validation Set**: A set of examples used to tune the parameters/model selection
  - **Test set**: A set of examples **only** to assess the performance of the fully trained model.
    - After assessing the final model with the test set, **YOU MUST NOT** further tune your model.



3-way split Scikit-learn

```
>>> from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.2)
```

Model and parameter selection should be done using X_train and X_val. Final testing should be done once on X_test.

Cross-Validation GridSearch Scikit-learn

```
>>> from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
>>> from sklearn.model_selection import GridSearchCV
>>> from sklearn.linear_model import SVC
>>> parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
>>> r = SVC()
>>> clf = GridSearchCV(r, parameters, scoring='f1_micro', cv=3)
>>> clf.fit(X_train, y_train) # Will perform 3-fold CV for each combination of parameters
```

GridSearchCV Attributes

```
>>> clf = GridSearchCV(r, parameters, scoring='f1_micro', cv=3)

>>> clf.fit(X_train, y_train)
```

To receive the best parameters found:

```
>>> print(clf.best_params_)
{'kernel':'rbf', 'C':1}
```

To get the best micro F1 score:

```
>>> print(clf.best_score_)
0.893
```

Exercise 3

Using the train/test iris dataset split from exercise 2. Train a model on the training dataset using GridSearchCV with the SVC kernel parameters "rbf" and "linear",and C parameters 0.001, 0.01, 0.1, 1., and 10. Print the training and validation scores for the best set of parameters.

Accuracy

- Assume we are classifying sentiment such that every review has 1 gold sentiment label (e.g., positive/negative).
- The classifier predicts 1 label for each review in the test set.
- Thus, every test set token has a predicted label (**pred**) and gold label (**gold**).
- The **accuracy** of our classifier is just the % of tokens for which the predicted label matched the gold label: (# **pred** = **gold**)/#reviews

Precision and Recall

- **Precision**: Measures how many things you predict as the positive class
  - Predict whether someone is depressed: classifier identifies EVERYONE as depressed
- **Recall**: Measures the number of things you do **NOT** predict.
  - Predict whether someone is depressed: classifier identifies NOBODY as depressed

Precision and Recall

Example: Predict whether someone is depressed given their social media data

True Positives (TP) = 6	**False Positives (FP) = 3**
False Negatives (FN) = 5	**True Negatives (TN) = 48**

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{6}{6+3} = 0.667$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{6}{6+5} = 0.545$$

F1

F1 $=$ harmonic mean of the precision and recall

$$\text{F1} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Intuition: Small precision and high recall (or vise-versa) will result in a small F1

Multiclass Evaluation

F1 is a metric for binary classification.

How can we use it for multi-class classification (e.g., positive, negative neutral)

Micro F1

Intuition: Sum all True Positives (TP), False Positives (FP), and False Negatives (FN) for every class, then calculate precision, recall, and f1 metrics

$$\text{Precision} = \frac{\sum_{i=1}^{K} TP_i}{\sum_{i=1}^{K} TP_i + FP_i}$$

$$\text{Recall} = \frac{\sum_{i=1}^{K} TP_i}{\sum_{i=1}^{K} TP_i + FN_i}$$

F1 is simply the harmonic mean of the above micro measures

Macro F1

$$\text{Precision}_i = \frac{TP_i}{TP_i + FP_i}$$

$$\text{Recall}_i = \frac{TP_i}{TP_i + FN_i}$$

$$\text{F1}_i = \frac{2 \times \text{precision}_i \times \text{recall}_i}{\text{precision}_i + \text{recall}_i}$$

$$\text{F1} = \frac{1}{K} \sum\nolimits_{i=1}^{K} F_i$$

Intuition: Average F1 for every class independently

Pros and Cons of Different Evaluation Metrics

- Accuracy and Micro F1 are sensitive to class distribution.
    - 100,000 class negative sentiment examples
    - 100 positive sentiment
    - Always predict negative $>$ 99% accuracy
- Macro F1, and F1 for binary classification, can evaluate classification with extreme imbalance
- So, which metric should you use? It depends.

How to generate features from text?

How can we transform the following sentences into vectors?

The cat is fat.

The dog is cute.

Text to Features

Define the set of features as the entire vocabulary: the, cat, is, fat, dog, cute

The cat is fat.

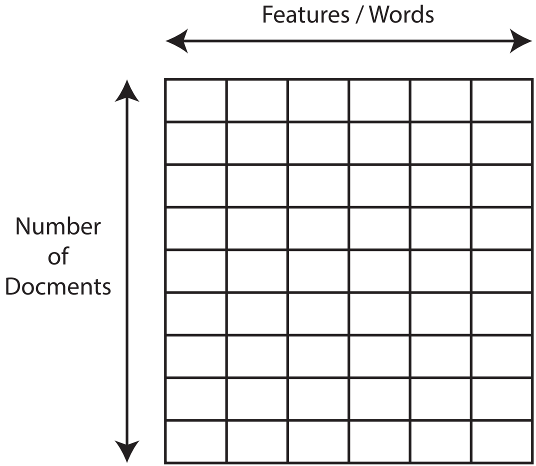
1	0	1	1	1	0
the	cute	fat	is	cat	dog

The dog is cute.

1	1	0	1	0	1
the	cute	fat	is	cat	dog

The document word matrix

$X =$



CountVectorizer

```
>>> txt = ["He is ::having a great Time, at the park time?",
    "She, unlike most women, is a big player on the park's grass.",
    "she can't be going"] # List of strings

>>> from sklearn.feature_extraction.text import CountVectorizer

>>> vec = CountVectorizer(ngram_range=(1, 1), min_df=1)

>>> cnt_stats = vec.fit(txt) # Create word/index dict

>>> bagOfWords = vec.transform(txt)

>>> print("Every feature:{}".format(vec.get_feature_names()))
Every feature:
['at', 'be', 'big', 'can', 'going', 'grass', 'great', 'having', 'he', 'is', 'most',
'on', 'park', 'player', 'she', 'the', 'time', 'unlike', 'women']
```

Scikit-learn Text

```
>>> print("Vocabulary size: {}".format(len(count_stats.vocabulary_)))
Vocabulary size: 10
```

vocabulary_ is a dictionary of word/index pairs. The index represents which column of the bagOfWords matrix is associated with each word

```
>>> print("Vocabulary content:{}".format(count_stats.vocabulary_))
{'he': 8, 'is': 9, 'having': 7, 'great': 6, 'time': 16, 'at': 0, 'the': 15, 'park': 12, 'she': 14, 'unlike': 17, 'most': 10, 'women': 18, 'big': 2, 'player': 13, 'on': 11, 'grass': 5, 'can': 3, 'be': 1, 'going': 4}
```

```
>>> bagOfWords # Will print a matrix. Not shown.
```

Scikit-learn Text: Unigrams and Bigrams

```
>>> vec = CountVectorizer(ngram_range=(1, 2), min_df=1,
stop_words='english')

>>> cnt_stats = vec.fit(txt) # Creates word/index dictionary for all
words in txt

>>> bagOfWords = vec.transform(txt) # Transform txt to a matrix

>>> print(vec.get_feature_names())
['big', 'big player', 'going', 'grass', 'great', 'great time', 'having', 'having
great', 'park', 'park grass', 'park time', 'player', 'player park', 'time', 'time
park', 'unlike', 'unlike women', 'women', 'women big']
```

Exercise 4

The tab separated file "sentiment-twitter-data.tsv" contains tweets annotated for sentiment. Load the data then do the following:

- create a bag of words feature representation for the tweets using CountVectorizer
- split it into a train/test split
- Use grid-search (CV) on the train split to find the best C parameters for an **LinearSVC** classifier (Do not use a standard SVC model because it will be to slow)
- Report (print) the accuracy of the final classifier on the test data.
- How many features were created with the bag of words representation (take the length of the the output from get_feature_names())?

End of Class

Post questions on Blackboard or email me.