# CS167: Machine Learning

## Evaluation Metrics

Tuesday, February 26th, 2023

Drake University

# Announcements

- Heads up: Quiz #1
  - due on Thursday 02/29 by 11:59pm
  - extended deadline by a day
- Notebook #3: Cross Validation
  - due Thursday 02/29 by 11:59pm
  - to submit, download the `ipynb` file from Colab

# Before we get started, let's load in our datasets:

- Make sure you change the path to match your Google Drive.
  - Load the `vehicle.csv` file from your Google Drive

```
[2]  #run this cell if you're using Colab:
     from google.colab import drive
     drive.mount('/content/drive')
```

```
#import the data:
#make sure the path on the line below corresponds to the path where you put your
import pandas as pd
import numpy as np
data = pd.read_csv('/content/drive/MyDrive/cs167_fall23/datasets/vehicles.csv')
pd.set_option('display.max_columns', 100)
iris = pd.read_csv('/content/drive/MyDrive/cs167_fall23/datasets/irisData.csv')
```

# Today’s Agenda

- Evaluation Metrics
  - Classification metrics
  - Regression metrics

# How do we know if our model is a 'good' model?

- We want to know how good our models are at making predictions... how can we test it? Examples:
  - what k-value should we use in kNN algorithm?
  - what is the effect on accuracy if I normalize the data?
  - should I use a weighted kNN algorithm or a normal kNN?

# Cross-Validation Code

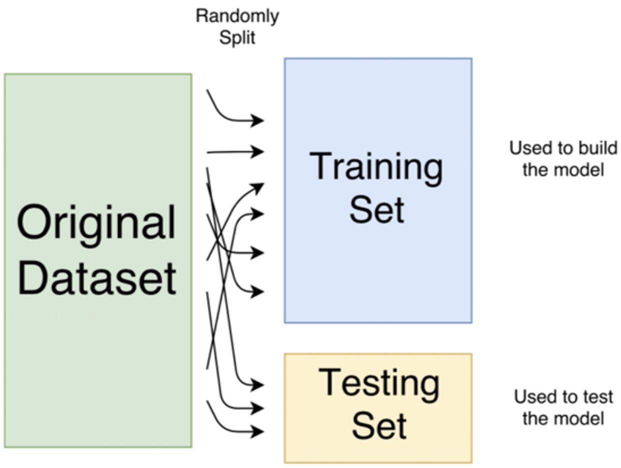
- A good rule of thumb is that we like to train our model with 80% of the given data examples (training set), and test it on 20% of the given data examples (testing set)

- Splitting datasets into training and testing sets with a Pandas DataFrame:

```
import pandas as pd
import numpy as np


#shuffle the iris "sampling" the full set in random order
shuffled_data = iris.sample(frac=1, random_state=41)

# set up training and testing set
test_data = shuffled_data.iloc[0:20] #test on the first 20 rows of shuffled
train_data = shuffled_data.iloc[20:] #train on the rest
train_data.shape
```



# Cross-Validation Metrics

- When doing cross-validation, how do we tell how well our model performed?

- How can we measure it?
    - depends on the task and what we want to know

- What's the difference between classification and regression?
    - The output variable in **classification** is categorical (or discrete)
    - The output variable in **regression** is numerical (or continuous)

# Today's Agenda

- Evaluation Metrics
  - Classification metrics
  - Regression metrics

# Classification metrics

- **Accuracy**: The fraction of test examples your model predicted correctly
  - *Example*: 17 out of 20 = 0.85 accuracy

- Issues with accuracy: suppose that a blood test for cancer has 99% accuracy
  - *can we safely assume this is a really good test?*
    - If the dataset is *unbalanced*, accuracy is not a reliable metric for the real performance of a classifier because it will yield misleading results
    - **Example**: Most people don't have cancer
  - Beware of what your metrics don't tell you

# Classification metrics

- **Accuracy**: The fraction of test examples your model predicted correctly
  - *Example*: 17 out of 20 = 0.85 accuracy

- Issues with accuracy: What about false negatives and false positives?
  - **false positives**: a test result which incorrectly indicates that a particular condition or attribute is present
  - **false negative**: a test result which incorrectly indicates that a particular condition or attribute is absent

| | | Predicted class | |
|---|---|---|---|
| | | *P* | *N* |
| Actual Class | *P* | 5 | 3 |
| | *N* | 2 | 90 |

# Classification metrics: Confusion Matrix

- **Confusion matrix:** A specific table layout that allows the visualization of the performance of an algorithm.
- Each row represents instances in an actual class
- While each column represents the instances in a predicted class
  - It makes it easy to see where your model is confusing the predicted and actual results. For a binary classification problem:

| | | Predicted condition | |
|---|---|---|---|
| | Total population = P + N | **Positive (PP)** | **Negative (PN)** |
| **Actual condition** | **Positive (P)** | **True positive (TP)** | **False negative (FN)** |
| **Actual condition** | **Negative (N)** | **False positive (FP)** | **True negative (TN)** |

# Classification metrics: Confusion Matrix

| | | Predicted condition | |
|---|---|---|---|
| | Total population = P + N | Positive (PP) | Negative (PN) |
| Actual condition | Positive (P) | True positive (TP) | False negative (FN) |
| | Negative (N) | False positive (FP) | True negative (TN) |

- **Confusion matrix:**
  - Each row represents instances in an actual class
  - While each column represents the instances in a predicted class

- To build the confusion matrix let's map the actual classifications and predicted classifications using the following flat table:

| Individual Number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Actual Classification | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Predicted Classification | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| Result | | | | | | | | | | | | |

# Exercise: Confusion Matrix

- **Confusion matrix:** A specific table layout that allows the visualization of the performance of an algorithm

| Individual Number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Actual Classification | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Predicted Classification | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| Result | | | | | | | | | | | | |

- Given the following confusion matrix:
  - how many true positive? 6
  - how many true negatives? 3
  - how many false positive? 1
  - how many false negatives? 2

# Exercise: Confusion Matrix

- **Confusion matrix:** A specific table layout that allows the visualization of the performance of an algorithm

| Individual Number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Actual Classification** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| **Predicted Classification** | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| **Result** | FN | FN | TP | TP | TP | TP | TP | TP | FP | TN | TN | TN |

- Given the following confusion matrix:
  - how many true positive? 6
  - how many true negatives? 3
  - how many false positive? 1
  - how many false negatives? 2

# Summarize the Results in `Confusion Matrix`

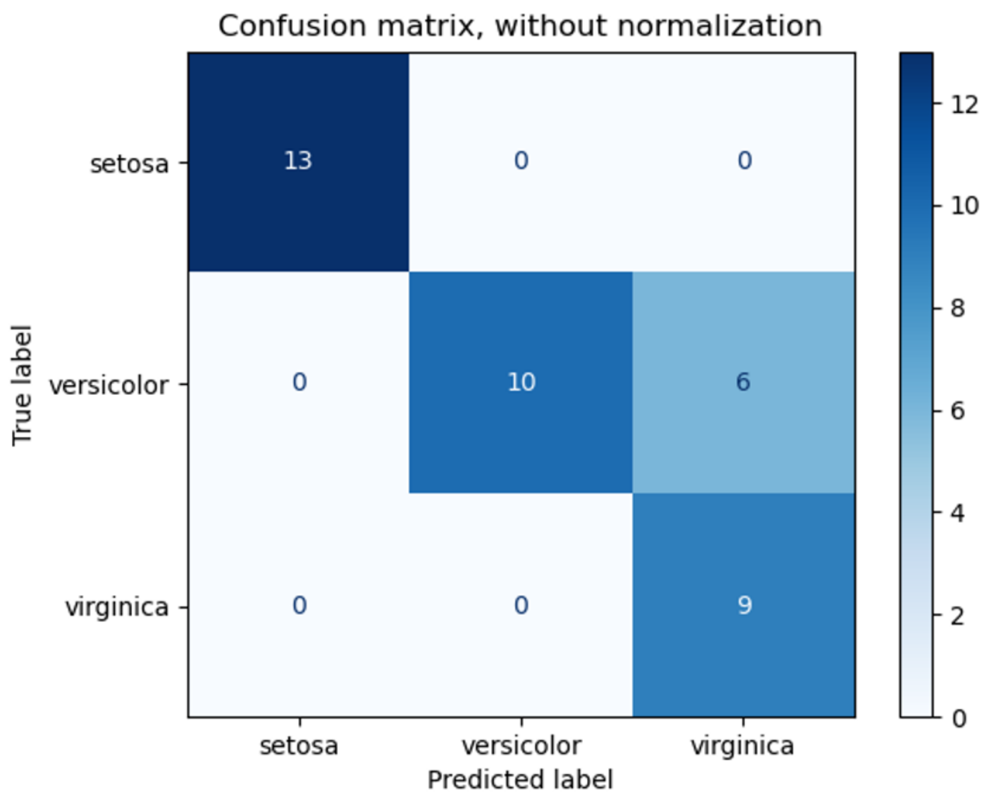
| | | Predicted condition | |
|---|---|---|---|
| | Total population = P + N | Positive (PP) | Negative (PN) |
| Actual condition | Positive (P) | True positive (TP) | False negative (FN) |
| | Negative (N) | False positive (FP) | True negative (TN) |

- Given the following confusion matrix:
  - how many true positive?
  - how many true negatives?
  - how many false positive?
  - how many false negatives?
  - what is the accuracy?

| | | Predicted class | |
|---|---|---|---|
| | | $P$ | $N$ |
| Actual Class | $P$ | 6 | 2 |
| | $N$ | 1 | 3 |

# Classification metrics: `Confusion Matrix`

- For a multi-class (more than 2) classification problem:
  - the confusion matrix looks like below where each row represents instances in an actual class; while each column represents the instances in a predicted class



# Today's Agenda

- Evaluation Metrics
  - Classification metrics
  - Regression metrics

# Regression metrics: Mean Absolute Error (MAE)

- **Mean Absolute Error (MAE):** the average difference ( absolute difference ie, always a positive value ) between the actual and predicted target values

$$\frac{\sum_{\text{test example } x_i} |actual(x_i) - predicted(x_i)|}{\text{number of test examples}}$$

**Example:**

| actual mpg | predicted mpg |
|---|---|
| 14 | 16 |
| 18 | 17 |
| 25 | 20 |

MAE:

$$(|14-16| + |18-17| + |25-20|)/3$$

$$= (2+1+5)/3 \approx 2.67$$

# Regression metrics: Mean Squared Error (MSE)

- **Mean Squared Error (MSE):** the average squared difference between the actual and predicted targets

$$\frac{\sum_{\text{test example } x_i} (actual(x_i) - predicted(x_i))^2}{\text{number of test examples}}$$

**Example:**

| actual mpg | predicted mpg |
|---|---|
| 14 | 16 |
| 18 | 17 |
| 25 | 20 |

MSE:

$$((14-16)^2+(18-17)^2+(25-20)^2)/3$$

$$= (2^2 + 1^2 + 5^2)/3 = 10$$

# Regression metrics: MAE vs. MSE

- **Mean Absolute Error (MAE):** the average difference ( absolute difference ie, always a positive value ) between the actual and predicted target values

- **Mean Squared Error (MSE):** the average squared difference between the actual and predicted targets

- What effect does the squaring have on the error measurements?

- Can you think of any scenarios where it might be better to use **`MAE`** over **`MSE`** or vis versa?

# Regression metrics: $R^2$

- Consider this naive prediction method: *“always predict the average target value.”*
  - Do you think this is a good predictor algorithm?
  - **No**

- So, we should be able to beat it — if we can't, we're in trouble. However, we can use this as a point of comparison.
  - An $R^2$ values of 0 means that you have done no better than the naive strategy of predicting the average.

```
from sklearn.metrics import r2_score
predictions= [12, 15.2, 21, 29]
actual = [14, 16, 19, 21]
r2 = r2_score(predictions, actual)
print(r2)
```

```
0.5652382092410821
```

# Interpreting $R^2$

- An $R^2$ values of 0 means that you have done no better than the naive strategy of predicting the average.

```
from sklearn.metrics import r2_score
predictions= [12, 15.2, 21, 29]
actual = [14, 16, 19, 21]
r2 = r2_score(predictions, actual)
print(r2)
```

```
0.5652382092410821
```

- Things you should know:
  - Usually **$R^2$** values fall between 0 and 1
  - 1 means you perfectly fit the data
  - 0 means you've done no better than average
  - Negative numbers mean that the naive model that predicts the average is actually a better predictor-- yours is really bad.

# Evaluation Metrics for kNN

- Let's see how accurate our kNN model is:
- Start with loading the data and setting up some cross-validation:

```
import pandas
data = pd.read_csv('/content/drive/MyDrive/cs167_fall23/datasets/irisData.csv')

#shuffle the data - "sampling" the full set in random order
shuffled_data = data.sample(frac=1, random_state=41)

#cross-validation
#use the first 20 rows in the shuffled set as testing data #train with the rest
test_data = shuffled_data.iloc[0:20]
train_data = shuffled_data.iloc[20:]
```

# Evaluation Metrics for kNN

- Then, let's bring in our `kNN()` function--here I'm calling it `classify_kNN()` because it uses `mode()` to return the prediction which only works for classification

```
def classify_kNN(new_example,train_data,k):
    #getting a copy of the training set just so we don't
    #mess up the original
    train_data_copy = train_data.copy()
    train_data_copy['distance_to_new'] = numpy.sqrt(
        (new_example['petal length']-train_data_copy['petal length'])**2
        +(new_example['sepal length']-train_data_copy['sepal length'])**2
        +(new_example['petal width']-train_data_copy['petal width'])**2
        +(new_example['sepal width']-train_data_copy['sepal width'])**2)

    sorted_data = train_data_copy.sort_values(['distance_to_new'])
    #mode to get most common thing in the first k examples in the sorted datafran
    #iloc to get the actual string, mode will return the string inside of a panda
    prediction = sorted_data.iloc[0:k]['species'].mode().iloc[0]
    return prediction
```

# Evaluation Metrics for kNN

- Now, let's write a function `classify_all_kNN(test_data, train_data, k)`
  - goes through each example in the `test_data`, and gets the prediction using our `classify_kNN()` function
  - It will return a pandas `Series` that has the predictions for each row in `test_data`
- It should look something like this:

```
def classify_all_kNN(test_data,train_data,k):
    #apply the classify_kNN function to each item in the test data with the trai
    #data and k passed as the other two arguments. The result will be a series o
    #the individual results.

    results = []

    for i in range(len(test_data)):
        prediction = classify_kNN(test_data.iloc[i], train_data,k)
        results.append(prediction)

    return pandas.Series(results)
```

# Evaluation Metrics for kNN

- Next, let's write a function for accuracy that will compare the actual species with the predicted species and return the percent we got correct

```
def calculate_accuracy(actual, predicted):
    #get the series comparing the two series: actual and predicted
    total_samples       = len(actual)                # Or since both series (actual and predicted)
    compared            = actual == predicted  # find which rows are of of the same value (eg, fo
    #print(compared)
    condition_correct   = compared == True

    correct_predictions = compared[ condition_correct ] # keep only those rows from 'compared'
    num_correct         = len(correct_predictions)      # count the size
    frac_correct        = num_correct/total_samples

    return frac_correct
```

# Evaluation Metrics for kNN

- Now, let's pull it all together and see how our kNN does:

```
import numpy

# Step 1: find the classification labels for all the samples in the test split using let's say k=11
k               = 11
predictionsKNN  = classify_all_kNN(test_data, train_data, k)

#Step 2: display the actual label and this will print out our predictions so we can see
print(f"{'ACTUAL':<{20}} {'PREDICTIONS':<{20}}")
for i in range(20):
  actual_sample     = test_data['species'].iloc[i]
  predicted_sample  = predictionsKNN.iloc[i]
  print(f"{actual_sample:<{20}} {predicted_sample:<{20}}")

# Step 3: calculate the evaluation metric 'accuracy' as we are doing ML classification
acc = calculate_accuracy(test_data['species'].reset_index(drop=True),
                         predictionsKNN.reset_index(drop=True))

print("accuracy:", acc)
```

# Evaluation Metrics for kNN

- Next, let's write a function for accuracy that will compare the actual species with the predicted species and return the percent we got correct

```
def calculate_accuracy(actual, predicted):
    #get the series comparing the two series: actual an
    total_samples       = len(actual)                  # O
    compared            = actual == predicted   # find wl
    #print(compared)
    condition_correct   = compared == True

    correct_predictions = compared[ condition_correct ]
    num_correct         = len(correct_predictions)
    frac_correct        = num_correct/total_samples

    return frac_correct
```

```
ACTUAL                PREDICTIONS
Iris-virginica        Iris-virginica
Iris-virginica        Iris-virginica
Iris-virginica        Iris-virginica
Iris-versicolor       Iris-versicolor
Iris-virginica        Iris-virginica
Iris-versicolor       Iris-versicolor
Iris-virginica        Iris-virginica
Iris-versicolor       Iris-versicolor
Iris-virginica        Iris-virginica
Iris-virginica        Iris-virginica
Iris-virginica        Iris-virginica
Iris-setosa           Iris-setosa
Iris-setosa           Iris-setosa
Iris-versicolor       Iris-versicolor
Iris-setosa           Iris-setosa
Iris-virginica        Iris-virginica
Iris-setosa           Iris-setosa
Iris-versicolor       Iris-virginica
Iris-setosa           Iris-setosa
Iris-setosa           Iris-setosa
accuracy: 0.95
```

# Evaluation Metrics for kNN

- You can visualize how the accuracies evolve for various *k=[1,3,5, …]*

```
import matplotlib.pyplot as plt
import pandas
# from sklearn.metrics import accuracy_score

#reload the data
data            = pd.read_csv('/content/drive/MyDrive/cs167_sp24/datasets/irisData.csv') #change this line 

# cross-validation to create 'train' and 'test' partitions
shuffled_data   = data.sample(frac=1, random_state = 41)
test_data       = shuffled_data.iloc[0:20]
train_data      = shuffled_data.iloc[20:]

# find the classification labels for all the samples in the test split using a series of k values
k_vals          = [1,3,5,9,15,21,31,51,101,129]
kNN_accuracies  = []

for k in k_vals:
    predictions       = classify_all_kNN(test_data,train_data,k)
    #acc              = accuracy_score(test_data['species'],predictions)
    acc               = calculate_accuracy(test_data['species'].reset_index(drop=True), predictions)
    kNN_accuracies.append(acc)

# plot the accuracy for each k-value to visualize the evolution of different parameter settings for k
# Use your code snippet from last week
plt.suptitle('Iris Data k-NN Experiment',fontsize=18)
plt.xlabel('k')
plt.ylabel('accuracy')
plt.plot(k_vals,kNN_accuracies,'ro-',label='k-NN')
plt.legend(loc='lower left', shadow=True)
plt.axis([0,130,0,1])

plt.show()
```

# Group Exercise#1: Implement a Regression Metric

- Write a function that takes in two Series and returns the **Mean Absolute Error (MAE):**

$$\frac{\sum_{\text{test example } x_i} |actual(x_i) - predicted(x_i)|}{\text{number of test examples}}$$

**Mean Absolute Error (MAE):** the average difference ( absolute difference ie, always a positive value ) between the actual and predicted target values

# Group Exercise#1: Implement a Regression Metric

```
[ ] import numpy as np
    np.absolute(-19)
```

```
19
```

```
[ ] def calculate_mae(actual, predicted):
        """
        takes in two Series of the same length, and returns the mean absolute error between the two series
        Hint: It's a lot simpler than you may think.
        """
        mean_abs_error = -1000
        #your code here


        return mean_abs_error
```

How can you test your code to make sure it's working correctly?

> Test your mae implementation using the following snippet

```
actual      = pd.Series(data=[10, 20, 30, 40, 50])
predicted   = pd.Series(data=[11, 22, 33, 44, 55])
mae         = calculate_mae(actual, predicted)
print(mae)
```

# Group Exercise#2: kNN Regression on Vehicle Dataset

- Write the function below:

```
def regress_kNN(new_example,train_data,k):
    #getting a copy of the training set just so we don't
    train_data_copy = train_data.copy()
    # copy the code from classify_kNN() and adjust it according to the new column names of the vehicle dataset

    return prediction
```

```
def regress_all_kNN(test_data,train_data, k):
    #apply the regress_kNN function to each item in the test data with the train
    #data and k passed as the other two arguments. The result will be a series of
    #the individual results.

    results = []

    for i in range(len(test_data)):
        prediction = regress_kNN(test_data.iloc[i], train_data,k)
        results.append(prediction)

    return pandas.Series(results)
```

# Group Exercise#2: kNN Regression on Vehicle Dataset

- Finish the Python code snippet below:

```
# do the regression experiment on 'vehicle' dataset

# step 1: cross-validation to create 'train' and 'test' partitions from vehicle
vehicle        = vehicle.sample(frac=1.0, random_state=3)
test_data      = vehicle.iloc[0:20]
train_data     = vehicle.iloc[20:]

# step 2: find the classification labels for all the samples in the test split using a series of k values
k_vals           = [1,3,5,9,15,21,31,51,101,129]
kNN_maes         = []

for k in k_vals:
    predictions          = regress_all_kNN(test_data,train_data,k)
    mae                  = calculate_mae(test_data['comb08'].reset_index(drop=True), predictions)
    kNN_maes.append(mae)
    print("mae=:", mae)

# step 3: plot the accuracy for each k-value to visualize the evolution of different parameter settings for k
# copy the code snippet from plot section above and adjust it by using the correct list
#plt.axis([0,130,0,1]) # HEADS-UP: FIX THE CORRECT RANGE
#plt.show()
```