CS167: Machine Learning

Evaluation Metrics
Evaluation Metric Coding with Graph Plot

Wednesday, February 25th, 2026

Review

- Evaluation Metrics
  - Classification metrics
  - Regression metrics
- Cross validation coding exercise
- Decision Tree

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

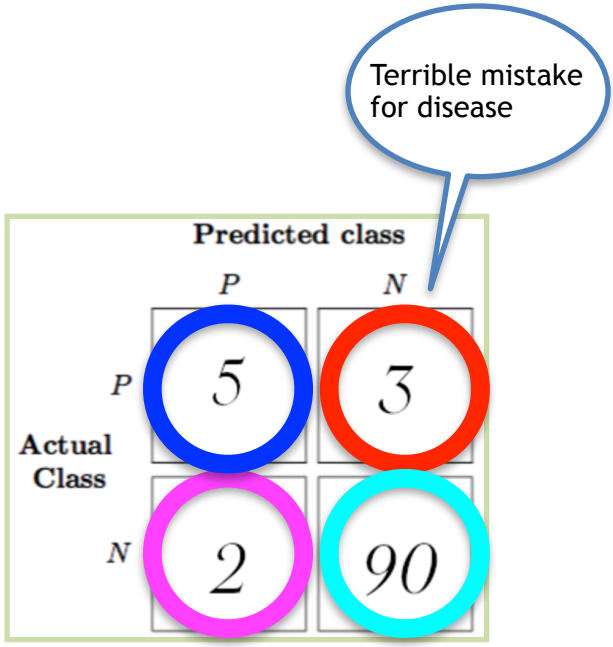
  - Beware of what your metrics don't tell you

Classification metrics

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

- **Correctly captures correct predictions:**
  - **True positives**: a test result which correctly indicates that a particular condition or attribute is present
  - **True negative**: a test result which correctly indicates that a particular condition or attribute is absent

- **Issues with accuracy:** What about
  - **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



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)
	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

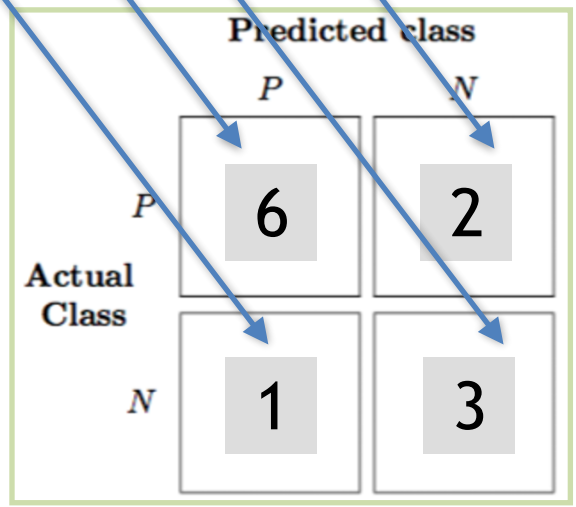
Individual Number	1	2	3	4	5	6	7	8	9	10	11	12
Actual Classification	1	1	1	1	1	1	1	1	0	0	0	0
Predicted Classification	0	0	1	1	1	1	1	1	1	0	0	0
Result												

- 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

Summarize the Results in Confusion Matrix

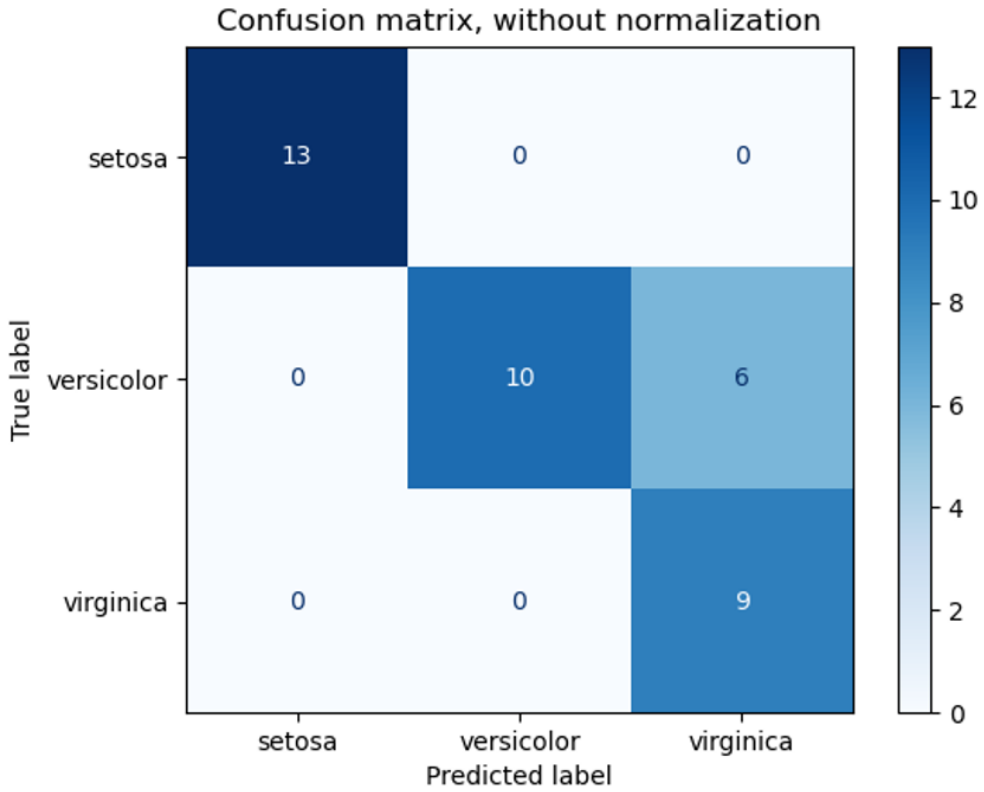
		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

- 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?



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



Review

- Evaluation Metrics
  - Classification metrics
  - Regression metrics
- Cross validation coding exercise
- Decision Tree

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.

Review

- Evaluation Metrics
  - Classification metrics
  - Regression metrics
- Cross validation coding exercise
- Decision Tree

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 as pd
import numpy as np
iris = pd.read_csv('/content/drive/MyDrive/cs167_sp26/datasets/irisData.csv')

#shuffle the iris "sampling" the full set in random order
shuffled_data = iris.sample(frac=1, random_state=41)
#shuffled_data = iris.sample(frac=1, random_state=10)

# set up training and testing set
number_of_test_samples = 20
iris_test     = shuffled_data.iloc[0:number_of_test_samples] #test on the first 20 rows of shuffled
iris_train    = shuffled_data.iloc[number_of_test_samples:] #train on the rest

predictors    = ['sepal length', 'sepal width', 'petal length', 'petal width']
target        = 'species'

# separating 'predictors' and 'target' for test split
test_data     = iris_test[predictors]
test_labels   = iris_test[target]

# separating 'predictors' and 'target' for train split
train_data    = iris_train[predictors]
train_labels  = iris_train[target]

print('number of test samples:', test_data.shape[0])
print('number of train samples:', train_data.shape[0])
```

```
number of test samples: 20
number of train samples: 130
```

Evaluation Metrics for kNN

- Then, let's bring in our `kNN()` function--here I'm calling it `classify_kNN()`

```
from sklearn.neighbors import KNeighborsClassifier

def knn_classifier(train_data, train_labels, test_data, k_value = 5):

  #---------------- STEP 2 ----------------
  #2. Create kNN classifier/regressor object
  my_knn = KNeighborsClassifier(n_neighbors=k_value)


  #---------------- STEP 3 ----------------
  #3. Call fit (to train the classification/regression model)
  my_knn.fit(train_data, train_labels)


  #---------------- STEP 4 ----------------
  #4. Call predict to generate predictions
  predictions = my_knn.predict(test_data)
  #print(f'Label of the new sample according to {k_value}-NN: {predictions[0]}')
  return my_knn, predictions
```

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(train_data, train_labels, test_data, k):
    #apply the classify_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(test_data.shape[0]):
        test_df = test_data.iloc[[i]] # double bracket to get a DataFrame (single bracket returns a Series)
        my_knn, prediction = knn_classifier(train_data, train_labels, test_df, k_value = k)
        results.append(prediction)

    return pd.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(train_data, train_labels, test_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_labels.iloc[i]
  predicted_sample  = predictionsKNN.iloc[i]
  print(f"{actual_sample:<{20}} {predicted_sample[0]:<{20}}")

# Step 3: calculate the evaluation metric 'accuracy' as we are doing ML classification
acc = calculate_accuracy(test_labels.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 w
    #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,…]*

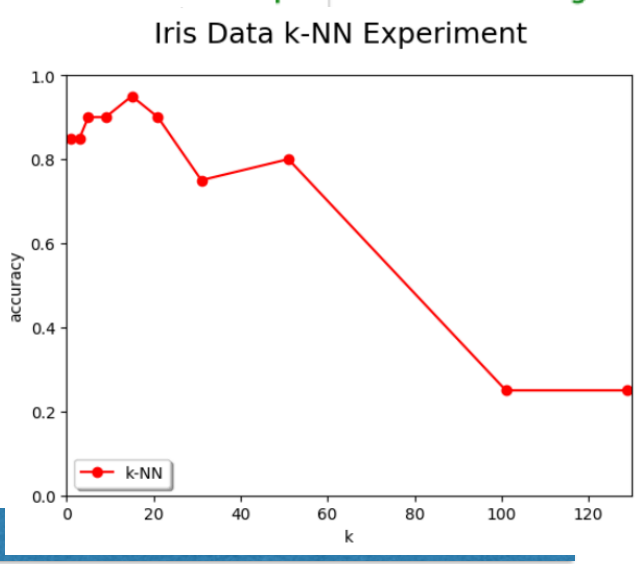
```
# find the classification labels for all the samples in the test split using a series of k values
k_vals          = [1,3,5,9,15,21,31,51,101,129]
kNN_accuracies  = []

for k in k_vals:
    predictionsKNN    = classify_all_kNN(train_data, train_labels, test_data, k)
    acc               = calculate_accuracy(test_labels.reset_index(drop=True),
                                           predictionsKNN.reset_index(drop=True))

    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

- Finish the missing sections of the two functions

```
from sklearn.neighbors import KNeighborsRegressor

def knn_regressor(train_data, train_labels, test_data, k_value = 5):
  # your code here
  # ...
  # ...
  # ...
  return my_knn, predictions
```

```
def regress_all_kNN(train_data, train_labels, test_data, k):
  results = []
  # your code here
  # ...
  # ...
  # ...

  return pd.Series(results)
```

Group Exercise#2: kNN Regression on Vehicle Dataset

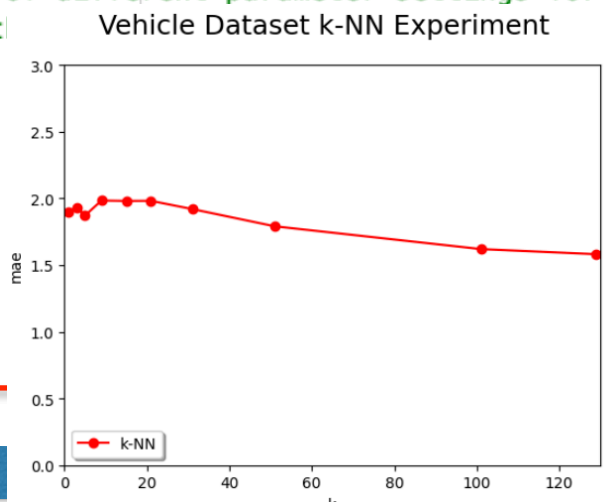
- Finish the Python code snippet below:

```
# 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:
    predictionsKNN    = regress_all_kNN(train_data, train_labels, test_data, k)
    mae               = calculate_mae(test_labels.reset_index(drop=True),
                                      predictionsKNN.reset_index(drop=True))
    kNN_maes.append(mae[0])
    print("mae=:", mae[0])
```

```
# 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 t
# your code here
# ...
# ...
# ...

plt.legend(loc='lower left', shadow=True)
plt.axis([0,130,0,1]) # HEADS-UP: FIX THE CORRECT RANGE
plt.show()
```



Announcements

- Heads up: Quiz #1
  - Will be released next week

- Notebook #3: Cross Validation
  - released
  - due next Wednesday March 4th, 2026 by 11:59pm
  - to submit, download the `ipynb` file from Colab