Classification I: training & predicting

Session learning objectives

By the end of the session, learners will be able to do the following:

- Recognize situations where a simple classifier would be appropriate for making predictions.
- Explain the K-nearest neighbor classification algorithm.
- Interpret the output of a classifier.
- Describe what a training data set is and how it is used in classification.
- Given a dataset with two explanatory variables/predictors, use K-nearest neighbor classification in Python using the `scikit-learn` framework to predict the class of a single new observation.

The classification problem

> predicting a categorical class (sometimes called a *label*) for an observation given its other variables (sometimes called *features*)

- Diagnose a patient as healthy or sick
- Tag an email as “spam” or “not spam”
- Predict whether a purchase is fraudulent

Training set

> Observations with known classes that we use as a basis for prediction

- Assign an observation without a known class (e.g., a new patient)
- To a class (e.g., diseased or healthy)

How?

- By similar it is to other observations for which we do know the class
  - (e.g., previous patients with known diseases and symptoms)

K-nearest neighbors

- One of many possible classification methods
  - KNN, decision trees, support vector machines (SVMs), logistic regression, neural networks, and more;

> Predict observations based on other observations “close” to it

Exploring a data set

Data:

- digitized breast cancer image features, created by Dr. William H. Wolberg, W. Nick Street, and Olvi L. Mangasarian
- Each row:
  - diagnosis (benign or malignant)
  - several other measurements (nucleus texture, perimeter, area, and more)
- Diagnosis for each image was conducted by physicians.

Formulate a predictive question:

> Can we use the tumor image measurements available to us to predict whether a future tumor image (with unknown diagnosis) shows a benign or malignant tumor?

Loading the cancer data

```
1 import pandas as pd
2 import altair as alt
3
4 cancer = pd.read_csv("data/wdbc.csv")
5 print(cancer)
```

```
           ID Class    Radius   Texture  Perimeter      Area  Smoothness  \
0      842302     M  1.096100 -2.071512   1.268817  0.983510    1.567087
1      842517     M  1.828212 -0.353322   1.684473  1.907030   -0.826235
2    84300903     M  1.578499  0.455786   1.565126  1.557513    0.941382
..        ...   ...       ...       ...        ...       ...         ...
566    926954     M  0.701667  2.043775   0.672084  0.577445   -0.839745
567    927241     M  1.836725  2.334403   1.980781  1.733693    1.524426
568     92751     B -1.806811  1.220718  -1.812793 -1.346604   -3.109349

     Compactness  Concavity  Concave_Points  Symmetry  Fractal_Dimension
0       3.280628   2.650542        2.530249  2.215566           2.253764
1      -0.486643  -0.023825        0.547662  0.001391          -0.867889
2       1.052000   1.362280        2.035440  0.938859          -0.397658
..           ...        ...             ...       ...                ...
566    -0.038646   0.046547        0.105684 -0.808406          -0.894800
567     3.269267   3.294046        2.656528  2.135315           1.042778
568    -1.149741  -1.113893       -1.260710 -0.819349          -0.560539

[569 rows x 12 columns]
```

> these values have been *standardized (centered and scaled)*

Describing the variables in the cancer data set

1. ID: identification number
2. Class: the diagnosis (M = malignant or B = benign)
3. Radius: the mean of distances from center to points on the perimeter
4. Texture: the standard deviation of gray-scale values
5. Perimeter: the length of the surrounding contour
6. Area: the area inside the contour
7. Smoothness: the local variation in radius lengths
8. Compactness: the ratio of squared perimeter and area
9. Concavity: severity of concave portions of the contour
10. Concave Points: the number of concave portions of the contour
11. Symmetry: how similar the nucleus is when mirrored
12. Fractal Dimension: a measurement of how “rough” the perimeter is

DataFrame; info

```
1  cancer.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   ID                  569 non-null    int64
 1   Class               569 non-null    object
 2   Radius              569 non-null    float64
 3   Texture             569 non-null    float64
 4   Perimeter           569 non-null    float64
 5   Area                569 non-null    float64
 6   Smoothness          569 non-null    float64
 7   Compactness         569 non-null    float64
 8   Concavity           569 non-null    float64
 9   Concave_Points      569 non-null    float64
 10  Symmetry            569 non-null    float64
 11  Fractal_Dimension   569 non-null    float64
dtypes: float64(10), int64(1), object(1)
memory usage: 53.5+ KB
```

Series; unique

```
1  cancer["Class"].unique()
```

```
array(['M', 'B'], dtype=object)
```

Series; replace

```
cancer["Class"] = cancer["Class"].replace({
    "M" : "Malignant",
    "B" : "Benign"
})

cancer["Class"].unique()
```

```
array(['Malignant', 'Benign'], dtype=object)
```

Exploring the cancer data

```
1  cancer.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   ID                  569 non-null    int64
 1   Class               569 non-null    object
 2   Radius              569 non-null    float64
 3   Texture             569 non-null    float64
 4   Perimeter           569 non-null    float64
 5   Area                569 non-null    float64
 6   Smoothness          569 non-null    float64
 7   Compactness         569 non-null    float64
 8   Concavity           569 non-null    float64
 9   Concave_Points      569 non-null    float64
 10  Symmetry            569 non-null    float64
 11  Fractal_Dimension   569 non-null    float64
dtypes: float64(10), int64(1), object(1)
memory usage: 53.5+ KB
```

```
1  cancer["Class"].value_counts()
```

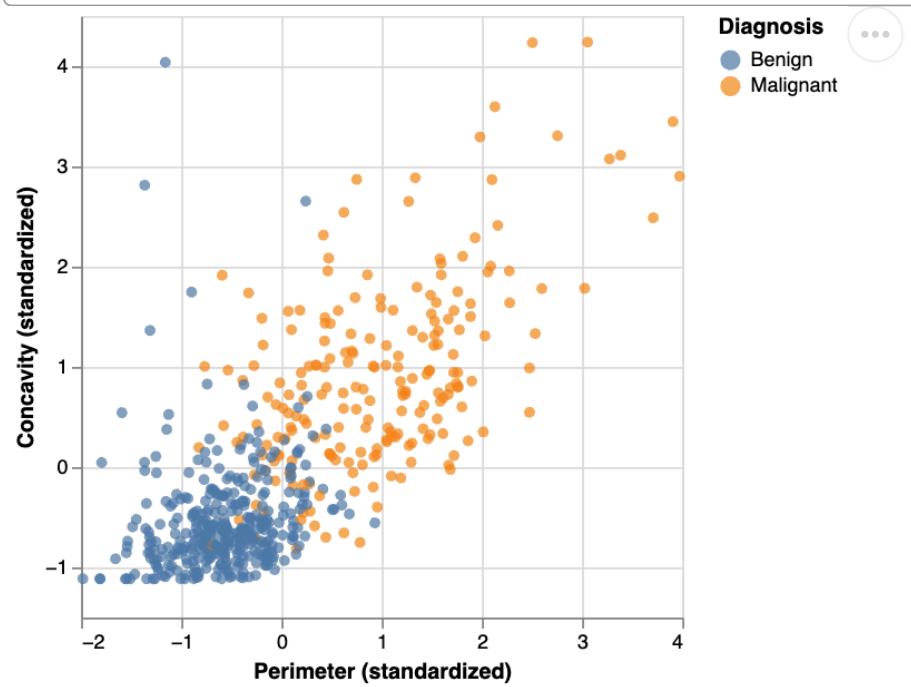
```
Class
Benign       357
Malignant    212
Name: count, dtype: int64
```

```
1  cancer["Class"].value_counts(normalize=True)
```

```
Class
Benign       0.627417
Malignant    0.372583
Name: proportion, dtype: float64
```

Visualization; scatter

```
perim_concav = alt.Chart(cancer).mark_circle().encode(
    x=alt.X("Perimeter").title("Perimeter (standardized)"),
    y=alt.Y("Concavity").title("Concavity (standardized)"),
    color=alt.Color("Class").title("Diagnosis")
)
perim_concav
```



- Malignant: upper right-hand corner
- Benign: lower left-hand corner

Classification with K-nearest neighbors

```
new_point = [2, 4]
attrs = ["Perimeter", "Concavity"]

points_df = pd.DataFrame(
    {"Perimeter": new_point[0], "Concavity": new_point[1], "Class": ["Unknown"]}
)

perim_concav_with_new_point_df = pd.concat((cancer, points_df), ignore_index=True)
print(perim_concav_with_new_point_df.iloc[[-1]])
```

```
     ID    Class  Radius  Texture  Perimeter  Area  Smoothness  Compactness  \
569 NaN  Unknown     NaN      NaN        2.0   NaN         NaN          NaN

     Concavity  Concave_Points  Symmetry  Fractal_Dimension
569        4.0             NaN       NaN                NaN
```

> Compute the distance matrix between each pair from a vector array X and Y

```
from sklearn.metrics.pairwise import euclidean_distances

# distance of new point to all other points
my_distances = euclidean_distances(perim_concav_with_new_point_df[attrs])[len(cancer)][:-1]
```

Distances (`euclidean_distances()`)

```
1 len(my_distances)
```

```
569
```

```
1 # distance of new point to all other points
2 my_distances
```

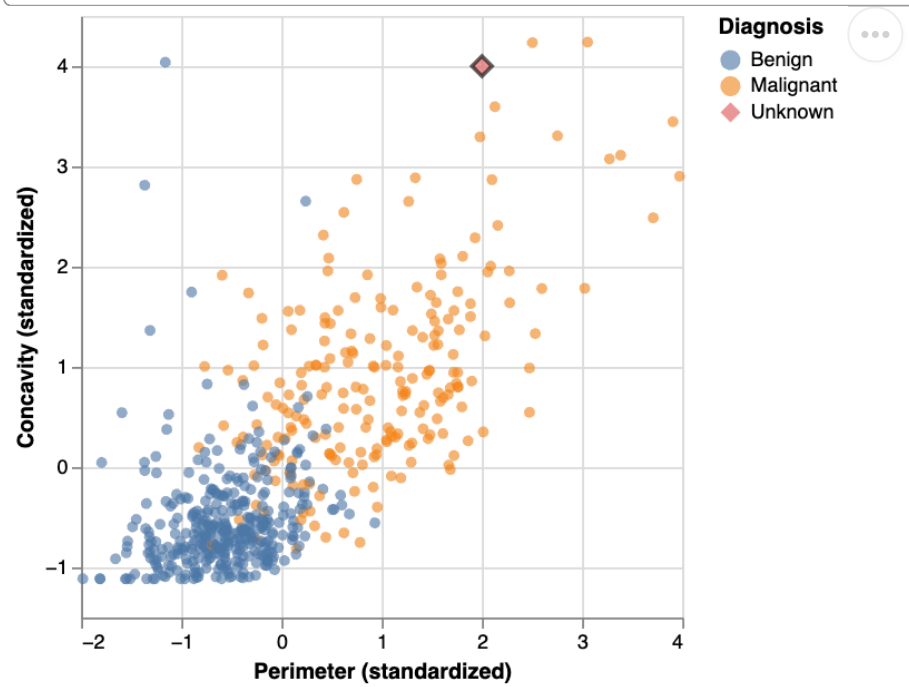
```
array([1.5348178 , 4.03617694, 2.67332814, 3.32713588, 2.63979916,
       3.93975767, 3.79946977, 4.45111808, 3.53679058, 3.24726821,
       4.951685  , 4.1538876 , 2.54585424, 4.15425044, 3.11637861,
       3.55044614, 4.59148187, 3.2419721 , 3.28753437, 4.80392314,
       5.0725937 , 5.77909213, 2.95751989, 3.74039071, 3.43925614,
       2.52875652, 3.77084705, 3.33265886, 3.38875698, 4.01548319,
       2.29844791, 4.41639587, 2.37500289, 3.07980178, 3.68423234,
       3.64538579, 3.96211289, 5.35324353, 5.15747872, 4.34753568,
       5.22721626, 4.73904767, 2.52193106, 4.45490282, 4.66689367,
       2.72752709, 6.12589249, 4.22308671, 4.99998799, 5.02870163,
       5.6046947 , 5.35214561, 5.55043261, 3.00901521, 4.79823955,
       5.48235796, 3.5095414 , 3.95297778, 5.63490804, 6.01390031,
       5.87933291, 5.91132619, 3.03871581, 5.49969602, 4.39948134,
       3.94696965, 5.86541424, 5.44232875, 3.5688574 , 5.25454378,
       3.82344861, 5.25379158, 3.18218244, 4.62286207, 5.32774445,
       4.16803695, 4.77170639, 3.116331  , 0.42493342, 5.19385448,
       5.30482795, 4.27283113, 1.5738716 , 2.72530186, 5.28695027,
       3.53135424, 4.07667556, 3.36046764, 5.02616064, 4.28105636,
       5.08987644, 4.06173178, 5.23583761, 5.12196626, 3.41101552,
       3.29340171, 5.45247337, 5.97564782, 5.33610215, 4.36938678,
       4.5747129 , 6.48184167, 5.47542661, 5.36604536, 5.613812  ,
       3.33949831, 5.01110652, 5.41189651, 0.55551675, 5.27565211,
```

K-nearest neighbors; classification

1. find the K "nearest" or "most similar" observations in our training set
2. predict new observation based on closest points

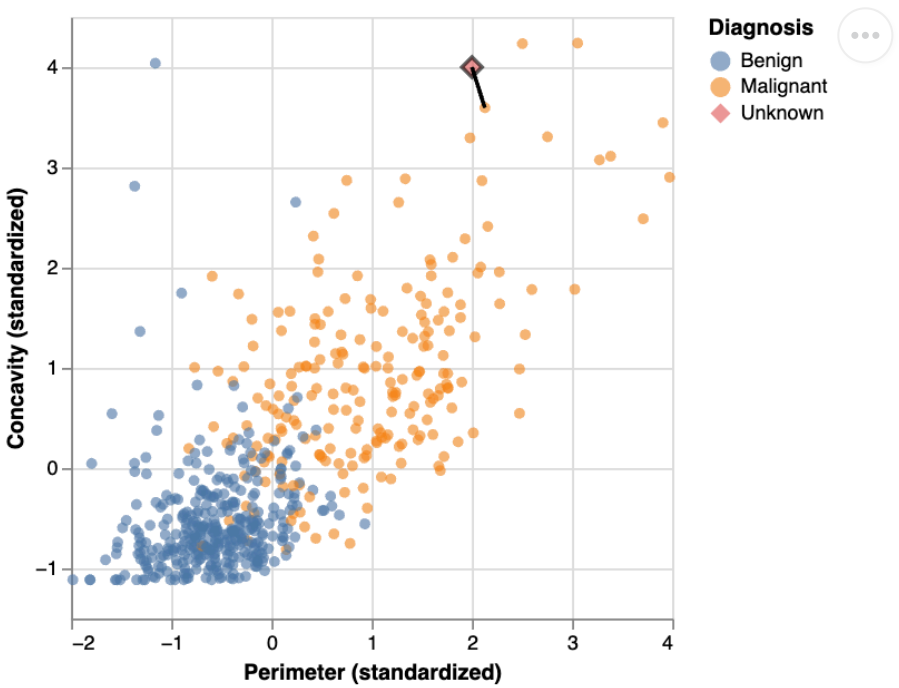
KNN Example: new point

```
perim_concav_with_new_point = (
    alt.Chart(perim_concav_with_new_point_df)
    .mark_point(opacity=0.6, filled=True, size=40)
    .encode(
        x=alt.X("Perimeter").title("Perimeter (standardized)"),
        y=alt.Y("Concavity").title("Concavity (standardized)"),
        color=alt.Color("Class").title("Diagnosis"),
        shape=alt.Shape("Class").scale(range=["circle", "circle", "diamond"]),
        size=alt.condition("datum.Class == 'Unknown'", alt.value(100), alt.value(30)),
        stroke=alt.condition("datum.Class == 'Unknown'", alt.value("black"), alt.value(None)),
    )
)

perim_concav_with_new_point
```

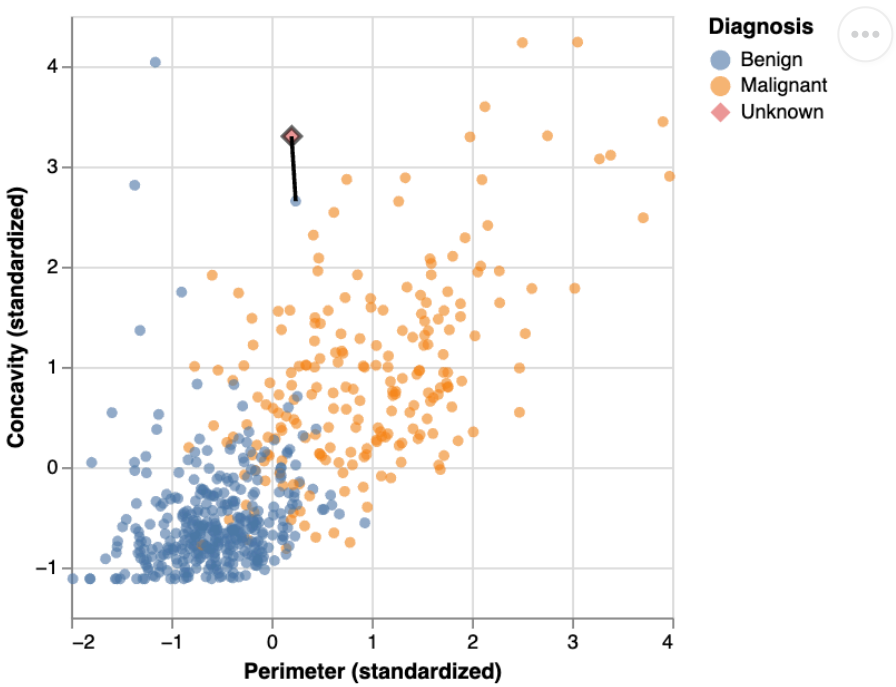


KNN example: closest point

> if a point is close to another in the scatter plot, then the perimeter and concavity values are similar, and so we may expect that they would have the same diagnosis

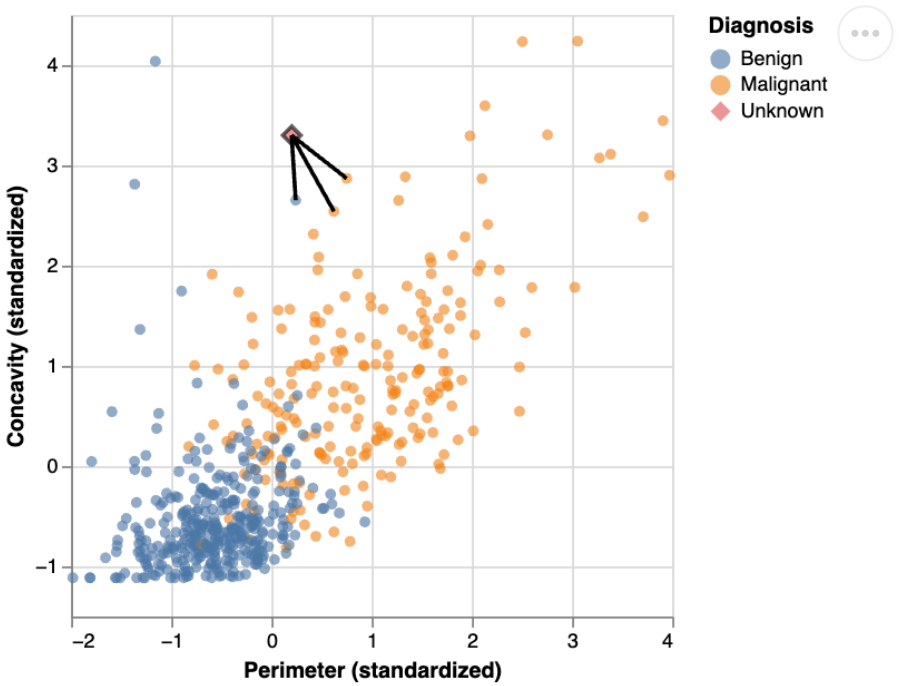


KNN Example: another new point



KNN: improve the prediction with k

we can consider several neighboring points, k=3

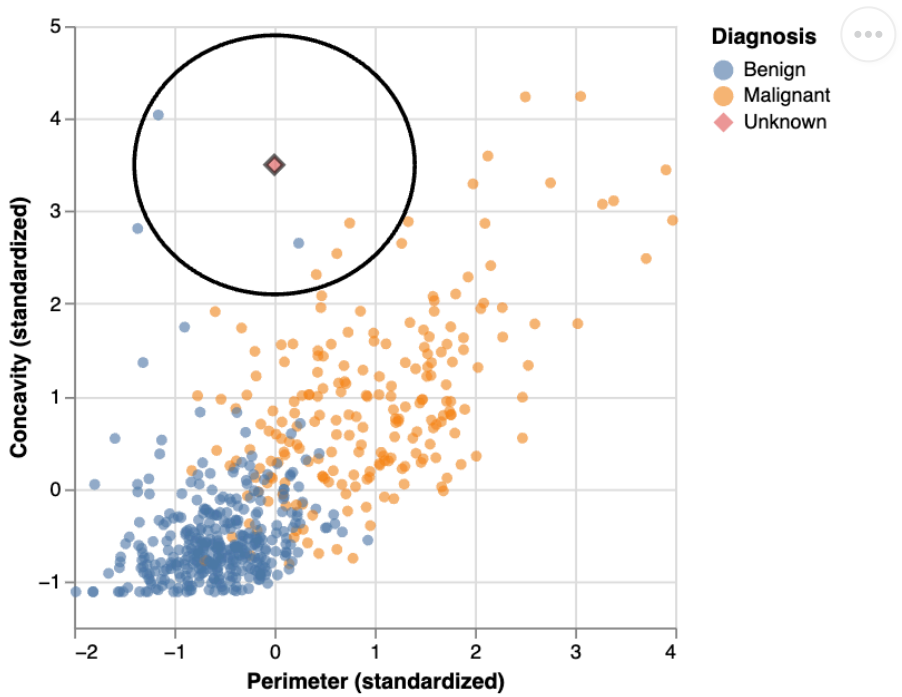


Distance between points

$$\text{Distance} = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}$$

Distance between points: k=5

> 3 of the 5 nearest neighbors to our new observation are malignant

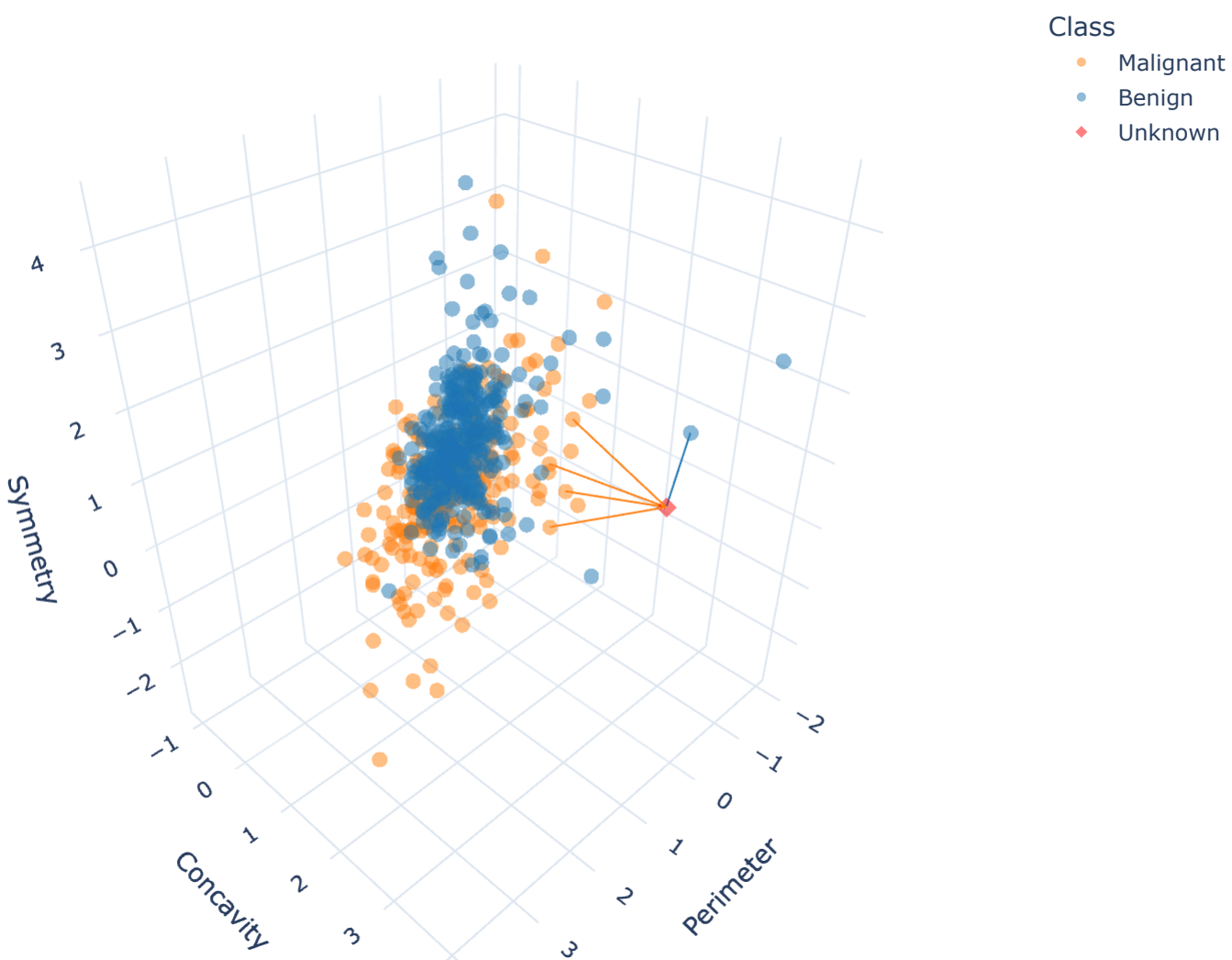


More than two explanatory variables: distance formula

The distance formula becomes

$$\text{Distance} = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_m - b_m)^2} \quad .$$

More than two explanatory variables: visualize



Summary of K-nearest neighbors algorithm

The K-nearest neighbors algorithm works as follows:

1. Compute the distance between the new observation and each observation in the training set
2. Find the K rows corresponding to the K smallest distances
3. Classify the new observation based on a majority vote of the neighbor classes

K-nearest neighbors with `scikit-learn`

- K-nearest neighbors algorithm is implemented in `scikit-learn`

```
from sklearn import set_config

# Output dataframes instead of arrays
set_config(transform_output="pandas")
```

Now we can get started with `sklearn` and `KNeighborsClassifier()`

```
from sklearn.neighbors import KNeighborsClassifier
```

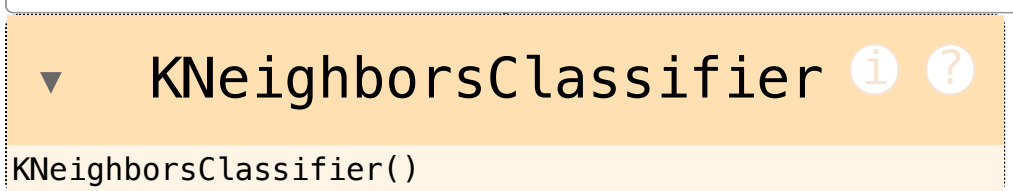
Review cancer data

```
1 cancer_train = cancer[["Class", "Perimeter", "Concavity"]]
2 print(cancer_train)
```

```
        Class  Perimeter  Concavity
0    Malignant   1.268817   2.650542
1    Malignant   1.684473  -0.023825
2    Malignant   1.565126   1.362280
..         ...        ...        ...
566  Malignant   0.672084   0.046547
567  Malignant   1.980781   3.294046
568     Benign  -1.812793  -1.113893

[569 rows x 3 columns]
```

scikit-learn: Create Model Object

```
1  from sklearn.neighbors import KNeighborsClassifier
2
3  knn = KNeighborsClassifier(n_neighbors=5)
4  knn
```



`scikit-learn`: Fit the model

```
knn.fit(
  X=cancer_train[["Perimeter", "Concavity"]],
  y=cancer_train["Class"]
)
```

KNeighborsClassifier

```
KNeighborsClassifier()
```

Note

1. We do not re-assign the variable
2. The arguments are X and y (note the capitialization). This comes from matrix notation.

scikit-learn: Predict

```
1  new_obs = pd.DataFrame({"Perimeter": [0], "Concavity": [3.5]})
2  print(new_obs)
```

```
   Perimeter  Concavity
0          0        3.5
```

```
1  knn.predict(new_obs)
```

```
array(['Malignant'], dtype=object)
```

Data preprocessing: Scaling

For KNN:

- the *scale* of each variable (i.e., its size and range of values) matters
- distance based algorithm

Compare these 2 scenarios:

- Person A (200 lbs, 6ft tall) vs Person B (202 lbs, 6ft tall)
- Person A (200 lbs, 6ft tall) vs Person B (200 lbs, 8ft tall)

All have a distance of 2

Data preprocessing: Centering

Many other models:

- *center* of each variable (e.g., its mean) matters as well
- Does not matter as much in KNN:
- Person A (200 lbs, 6ft tall) vs Person B (202 lbs, 6ft tall)
- Person A (200 lbs, 6ft tall) vs Person B (200 lbs, 8ft tall)

Difference in weight is in the 10s, difference in height is fractions of a foot.

Data preprocessing: Standardization

- The mean is used to center, the standard deviation is used to scale
- Standardization: transform the data such that the mean is 0, and a standard deviation is 1

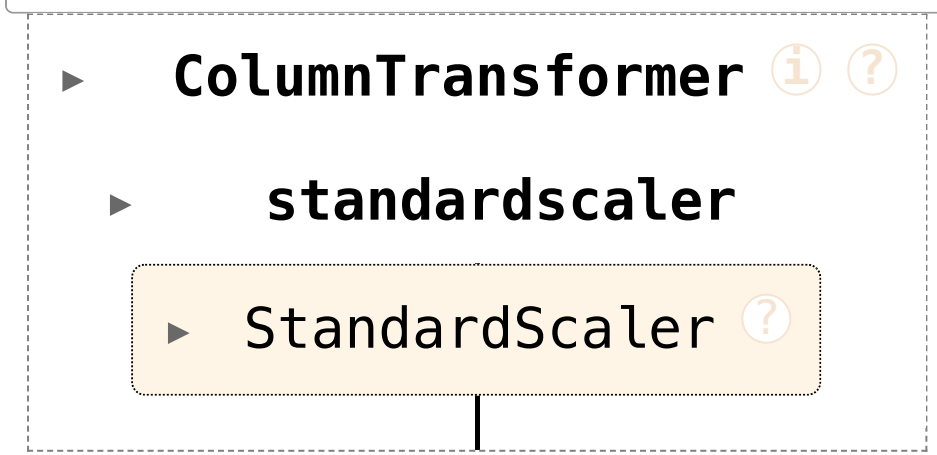
```
unscaled_cancer = pd.read_csv("data/wdbc_unscaled.csv")[["Class", "Area", "Smoothness"]]
unscaled_cancer["Class"] = unscaled_cancer["Class"].replace({
   "M" : "Malignant",
   "B" : "Benign"
})
unscaled_cancer
```

	Class	Area	Smoothness
0	Malignant	1001.0	0.11840
1	Malignant	1326.0	0.08474
2	Malignant	1203.0	0.10960
...	...	...	...
566	Malignant	858.1	0.08455
567	Malignant	1265.0	0.11780
568	Benign	181.0	0.05263

scikit-learn: ColumnTransformer

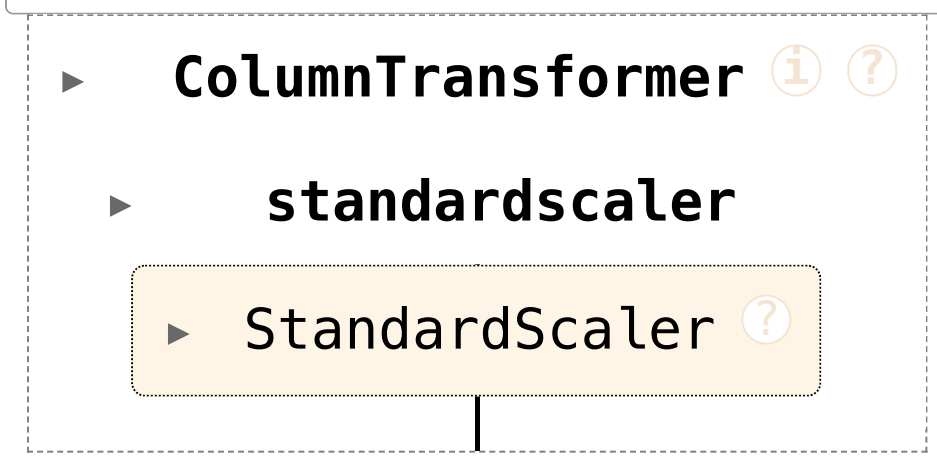
- scikit-learn has a preprocessing module
  - StandardScaler(): scale our data
- make_column_transformer: creates a ColumnTransformer to select columns

```
from sklearn.preprocessing import StandardScaler
from sklearn.compose import make_column_transformer

preprocessor = make_column_transformer(
    (StandardScaler(), ["Area", "Smoothness"]),
)
preprocessor
```



scikit-learn: Select numeric columns

```
from sklearn.compose import make_column_selector

preprocessor = make_column_transformer(
    (StandardScaler(), make_column_selector(dtype_include="number")),
)
preprocessor
```



scikit-learn: transform

Scale the data

```
1 preprocessor.fit(unscaled_cancer)
2 scaled_cancer = preprocessor.transform(unscaled_cancer)
```

Compare unscaled vs scaled

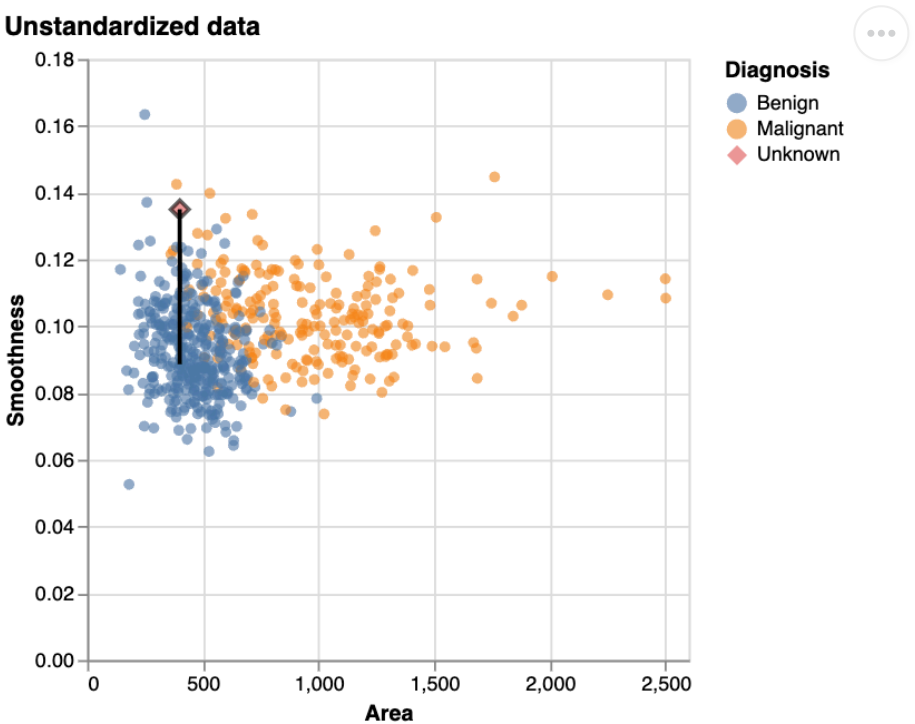
```
1 print(unscaled_cancer)
```

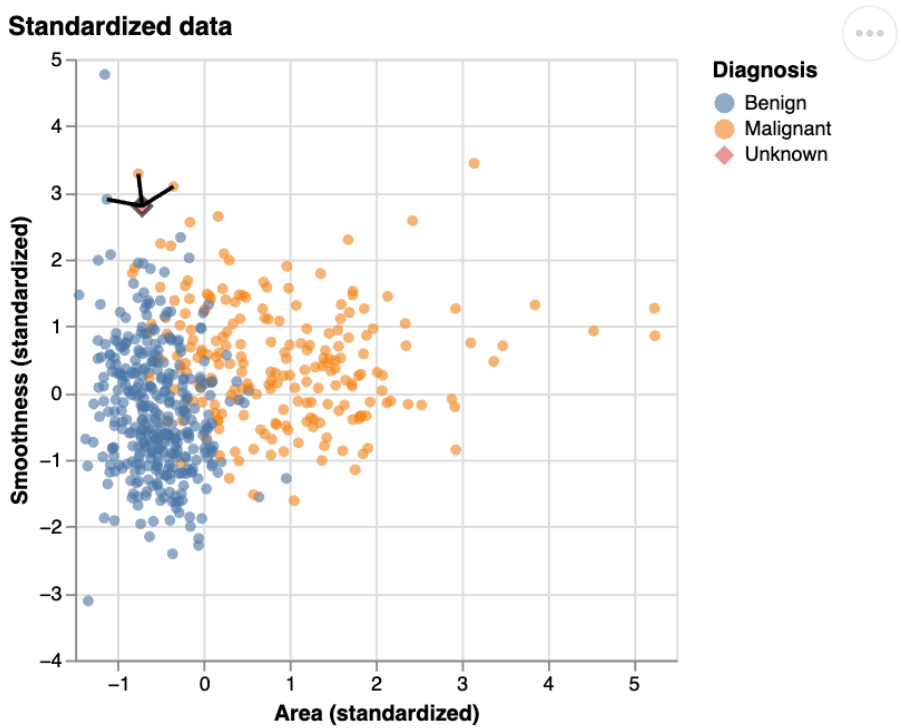
```
         Class    Area  Smoothness
0    Malignant  1001.0     0.11840
1    Malignant  1326.0     0.08474
2    Malignant  1203.0     0.10960
..         ...     ...         ...
566  Malignant   858.1     0.08455
567  Malignant  1265.0     0.11780
568     Benign   181.0     0.05263

[569 rows x 3 columns]
```

```
1 print(scaled_cancer)
```

```
     standardscaler__Area  standardscaler__Smoothness
0                0.984375                    1.568466
1                1.908708                   -0.826962
2                1.558884                    0.942210
..                    ...                         ...
566              0.577953                   -0.840484
567              1.735218                    1.525767
568             -1.347789                   -3.112085

[569 rows x 2 columns]
```

Visualize unstandarized vs standarized data





Why `scikit-learn` pipelines?

- Manually standarizing is error prone
- Does not automatically account for new data
- Prevent data leakage by processing on training data to use on test data (later)
- Need same mean and standarization from training to use on test / new data

Balancing + class imbalance

What if we have class imbalance? i.e., if the response variable has a big difference in frequency counts between classes?

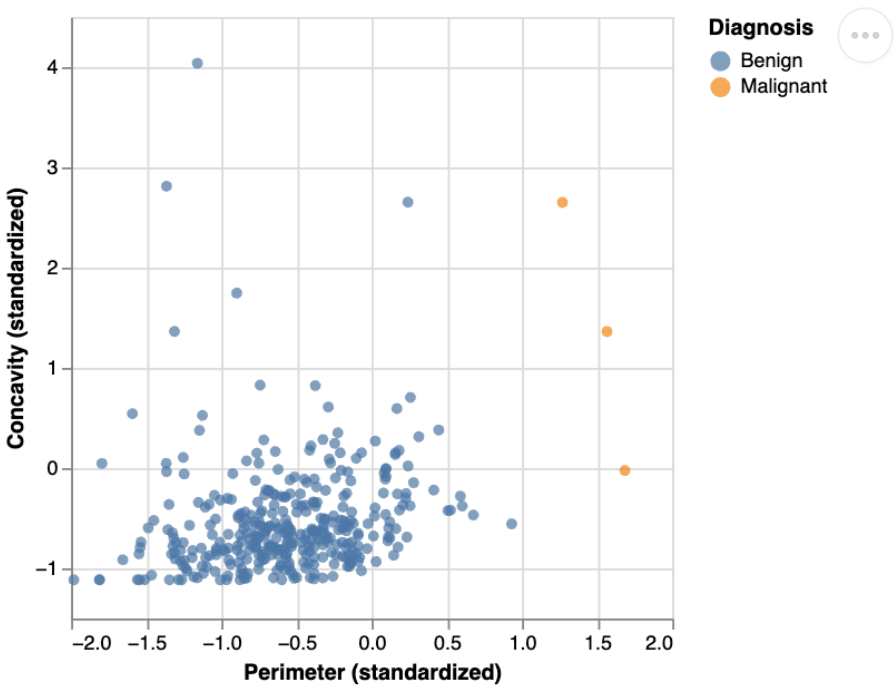
```
rare_cancer = pd.concat((
    cancer[cancer["Class"] == "Benign"],
    cancer[cancer["Class"] == "Malignant"].head(3) # only 3 total
))
print(rare_cancer)
```

```
          ID      Class    Radius   Texture  Perimeter      Area  Smoothness  \
19   8510426     Benign -0.166653 -1.146154  -0.185565 -0.251735    0.101657
20   8510653     Benign -0.297184 -0.832276  -0.260877 -0.383301    0.792066
21   8510824     Benign -1.311926 -1.592558  -1.301661 -1.082620    0.429441
..       ...        ...       ...       ...        ...       ...         ...
0     842302  Malignant  1.096100 -2.071512   1.268817  0.983510    1.567087
1     842517  Malignant  1.828212 -0.353322   1.684473  1.907030   -0.826235
2   84300903  Malignant  1.578499  0.455786   1.565126  1.557513    0.941382

    Compactness  Concavity  Concave_Points  Symmetry  Fractal_Dimension  \
19    -0.436466  -0.277965       -0.028584  0.267676          -0.727669
20     0.429044  -0.540886       -0.459223  0.566790           0.752425
21    -0.746429  -0.743094       -0.725698  0.012334           0.885562
..          ...        ...             ...       ...                ...
0      3.280628   2.650542        2.530249  2.215566           2.253764
1     -0.486643  -0.023825        0.547662  0.001391          -0.867889
2      1.052000   1.362280        2.035440  0.938859          -0.397658

    dist_from_new
19       3.852759
20       4.072405
21       4.546829
```

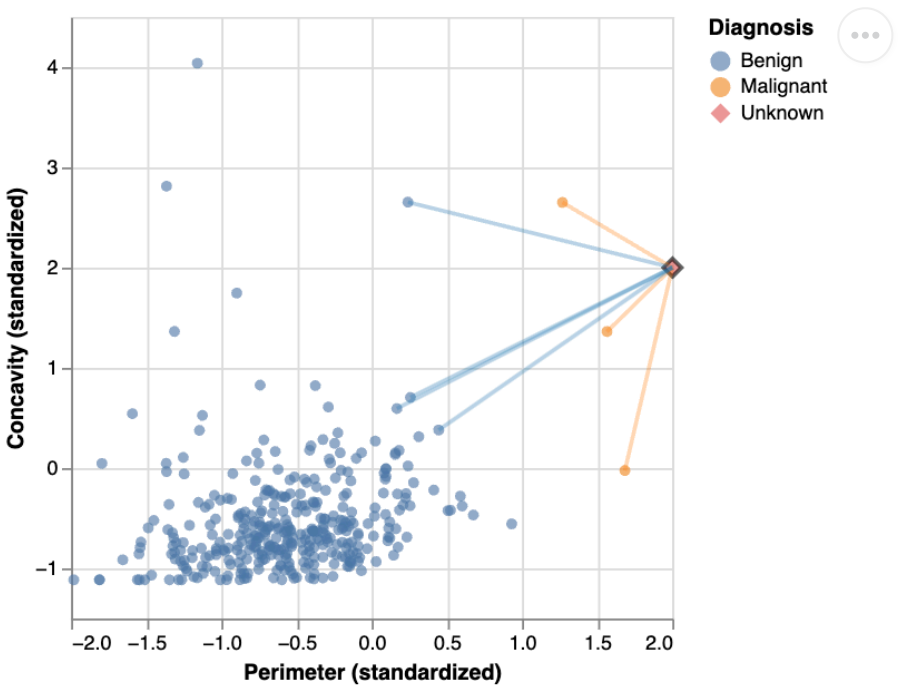
Visualizing class imbalance

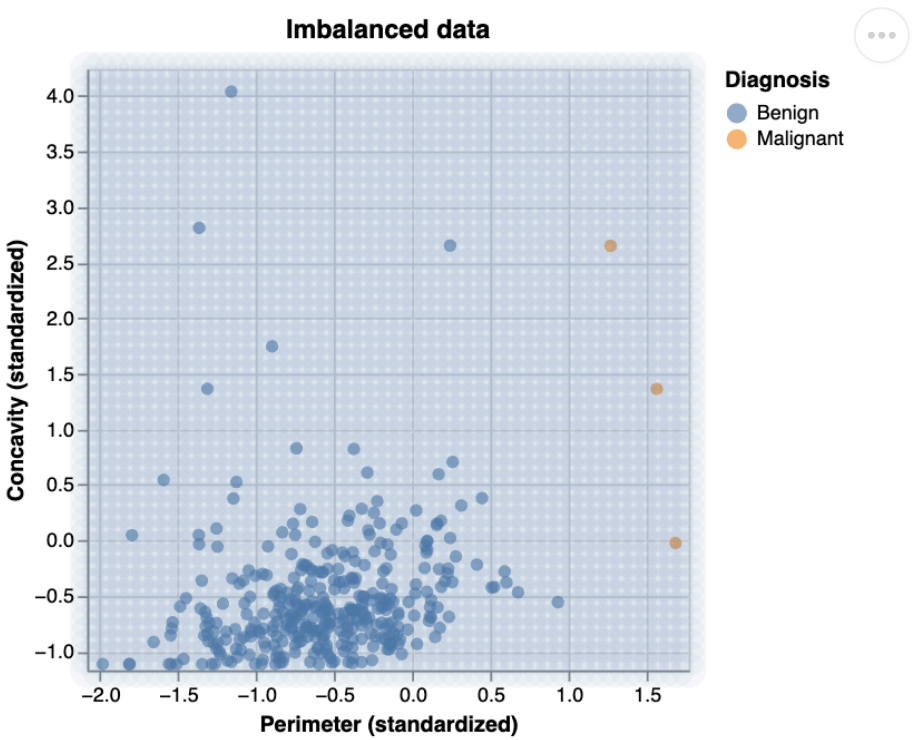
```
1 rare_cancer["Class"].value_counts()
```

```
Class
Benign       357
Malignant      3
Name: count, dtype: int64
```



Predicting with class imbalance





Upsampling

Rebalance the data by *oversampling* the rare class

1. Separate the classes out into their own data frames by filtering
2. Use the `.sample()` method on the rare class data frame
    - Sample with replacement so the classes are the same size
3. Use the `.value_counts()` method to see that our classes are now balanced

Upsampling: code

Set seed

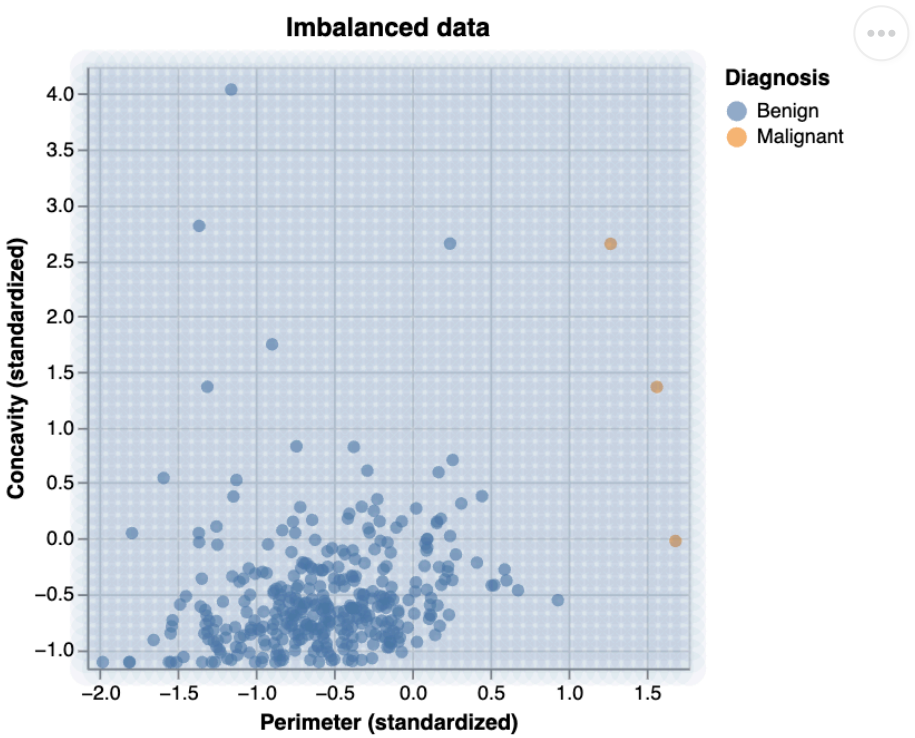
```
1  import numpy as np
2
3  np.random.seed(42)
```

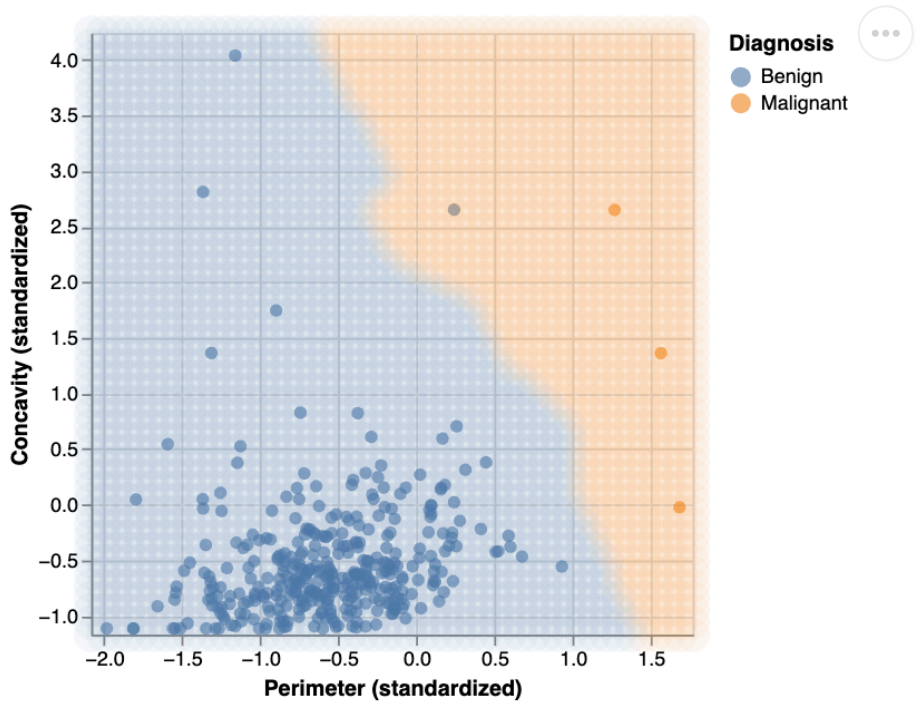
Upsample the rare class

```
1  malignant_cancer = rare_cancer[rare_cancer["Class"] == "Malignant"]
2  benign_cancer = rare_cancer[rare_cancer["Class"] == "Benign"]
3  malignant_cancer_upsample = malignant_cancer.sample(
4      n=benign_cancer.shape[0], replace=True
5  )
6  upsampled_cancer = pd.concat((malignant_cancer_upsample, benign_cancer))
7  upsampled_cancer["Class"].value_counts()
```

```
Class
Malignant    357
Benign       357
Name: count, dtype: int64
```

Upsampling: Re-train KNN k=7





Missing data

Assume we are only looking at “randomly missing” data

```
missing_cancer = pd.read_csv("data/wdbc_missing.csv")[
    ["Class", "Radius", "Texture", "Perimeter"]
]
missing_cancer["Class"] = missing_cancer["Class"].replace(
    {"M": "Malignant", "B": "Benign"}
)
print(missing_cancer)
```

```
       Class    Radius   Texture  Perimeter
0  Malignant       NaN       NaN   1.268817
1  Malignant  1.828212 -0.353322   1.684473
2  Malignant  1.578499       NaN   1.565126
3  Malignant -0.768233  0.253509  -0.592166
4  Malignant  1.748758 -1.150804   1.775011
5  Malignant -0.475956 -0.834601  -0.386808
6  Malignant  1.169878  0.160508   1.137124
```

Missing data: `.dropna()`

KNN computes distances across all the features, it needs complete observations

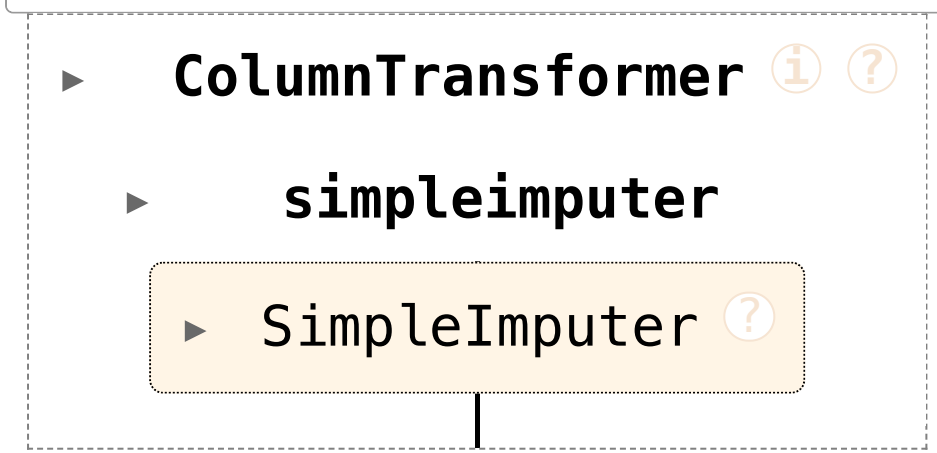
```
# drop incomplete observations
no_missing_cancer = missing_cancer.dropna()
print(no_missing_cancer)
```

```
      Class    Radius   Texture  Perimeter
1  Malignant  1.828212 -0.353322   1.684473
3  Malignant -0.768233  0.253509  -0.592166
4  Malignant  1.748758 -1.150804   1.775011
5  Malignant -0.475956 -0.834601  -0.386808
6  Malignant  1.169878  0.160508   1.137124
```

Missing data: `SimpleImputer()`

We can impute missing data (with the mean) if there's too many missing values

```
from sklearn.impute import SimpleImputer

preprocessor = make_column_transformer(
    (SimpleImputer(), ["Radius", "Texture", "Perimeter"]),
    verbose_feature_names_out=False,
)
preprocessor
```



Imputed data

```
preprocessor.fit(missing_cancer)
imputed_cancer = preprocessor.transform(missing_cancer)
```

```
print(missing_cancer)
```

	Class	Radius	Texture	Perimeter
0	Malignant	NaN	NaN	1.268817
1	Malignant	1.828212	-0.353322	1.684473
2	Malignant	1.578499	NaN	1.565126
3	Malignant	-0.768233	0.253509	-0.592166
4	Malignant	1.748758	-1.150804	1.775011
5	Malignant	-0.475956	-0.834601	-0.386808
6	Malignant	1.169878	0.160508	1.137124

```
print(imputed_cancer)
```

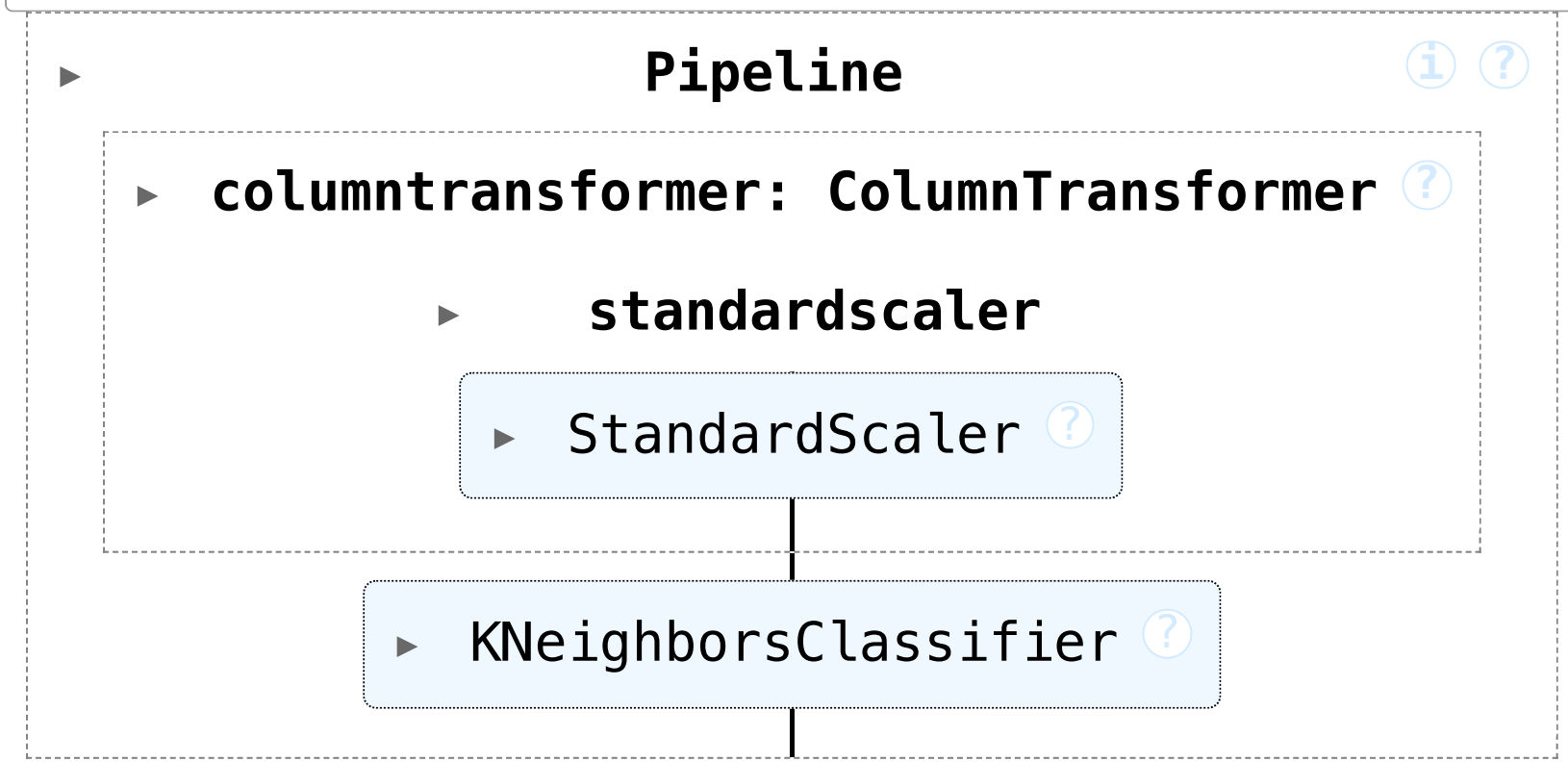
	Radius	Texture	Perimeter
0	0.846860	-0.384942	1.268817
1	1.828212	-0.353322	1.684473
2	1.578499	-0.384942	1.565126
3	-0.768233	0.253509	-0.592166
4	1.748758	-1.150804	1.775011
5	-0.475956	-0.834601	-0.386808
6	1.169878	0.160508	1.137124

Put it all together: Preprocessor

```
# load the unscaled cancer data, make Class readable
unscaled_cancer = pd.read_csv("data/wdbc_unscaled.csv")
unscaled_cancer["Class"] = unscaled_cancer["Class"].replace(
    {"M": "Malignant", "B": "Benign"}
)

# create the K-NN model
knn = KNeighborsClassifier(n_neighbors=7)

# create the centering / scaling preprocessor
preprocessor = make_column_transformer(
    (StandardScaler(), ["Area", "Smoothness"]),
    # more column transformers here
)
```

Put it all together: Pipeline

```
from sklearn.pipeline import make_pipeline

knn_pipeline = make_pipeline(preprocessor, knn)
knn_pipeline.fit(
    X=unscaled_cancer,
    y=unscaled_cancer["Class"]
)
knn_pipeline
```



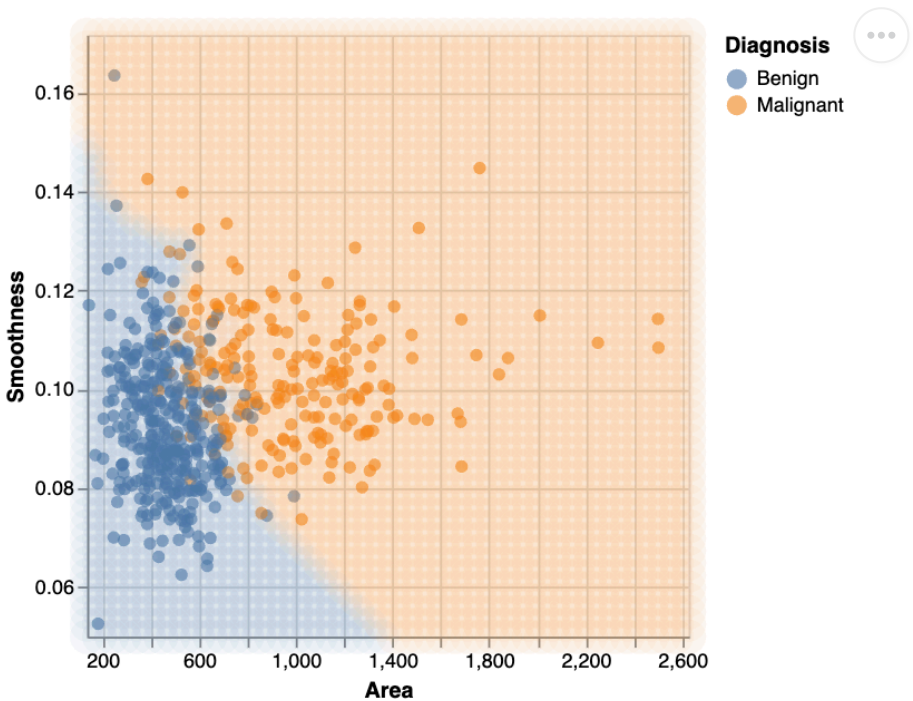
Put it all together: Predict

```
new_observation = pd.DataFrame(
    {"Area": [500, 1500], "Smoothness": [0.075, 0.1]}
)
prediction = knn_pipeline.predict(new_observation)
prediction
```

```
array(['Benign', 'Malignant'], dtype=object)
```

Prediction Area

Model prediction area.



- Points are on original unscaled data
- Area is using the pipeline model

Reference Code

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn.compose import (
    make_column_transformer,
)


# load the unscaled cancer data
unscaled_cancer = pd.read_csv(
    "data/wdbc_unscaled.csv"
)

# make Class readable
unscaled_cancer["Class"] = unscaled_cancer[
    "Class"
].replace({"M": "Malignant", "B": "Benign"})
```

```
# create the K-NN model
knn = KNeighborsClassifier(n_neighbors=7)

# create the centering / scaling preprocessor
preprocessor = make_column_transformer(
    (StandardScaler(), ['Area', 'Smoothness']),
    # more column transformers here
)

knn_pipeline = make_pipeline(preprocessor, knn)
knn_pipeline.fit(X=unscaled_cancer, y=unscaled_can
knn_pipeline

new_observation = pd.DataFrame(
    {
        'Area': [500, 1500],
        'Smoothness': [0.075, 0.1],
    }
)
prediction = knn_pipeline.predict(new_observation)
prediction
```

```
array(['Benign', 'Malignant'], dtype=object)
```

Additional resources

- The Classification I: training & predicting chapter of Data Science: A First Introduction (Python Edition) by Tiffany Timbers, Trevor Campbell, Melissa Lee, Joel Ostblom, Lindsey Heagy contains all the content presented here with a detailed narrative.
- The `scikit-learn` website is an excellent reference for more details on, and advanced usage of, the functions and packages in this lesson. Aside from that, it also offers many useful tutorials to get you started.
- *An Introduction to Statistical Learning* by Gareth James Daniela Witten Trevor Hastie, and Robert Tibshirani provides a great next stop in the process of learning about classification. Chapter 4 discusses additional basic techniques for classification that we do not cover, such as logistic regression, linear discriminant analysis, and naive Bayes.

References

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In ECML PKDD Workshop: Languages for Data Mining and Machine Learning, 108–122. 2013.

Thomas Cover and Peter Hart. Nearest neighbor pattern classification. IEEE Transactions on Information Theory, 13(1):21–27, 1967.

Evelyn Fix and Joseph Hodges. Discriminatory analysis. nonparametric discrimination: consistency properties. Technical Report, USAF School of Aviation Medicine, Randolph Field, Texas, 1951.

William Nick Street, William Wolberg, and Olvi Mangasarian. Nuclear feature extraction for breast tumor diagnosis. In International Symposium on Electronic Imaging: Science and Technology. 1993.

Stanford Health Care. What is cancer? 2021. URL: https://stanfordhealthcare.org/medical-conditions/cancer/cancer.html.