Model Engineering

COSC 423/523: Artificial Intelligence
Fall 2021

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

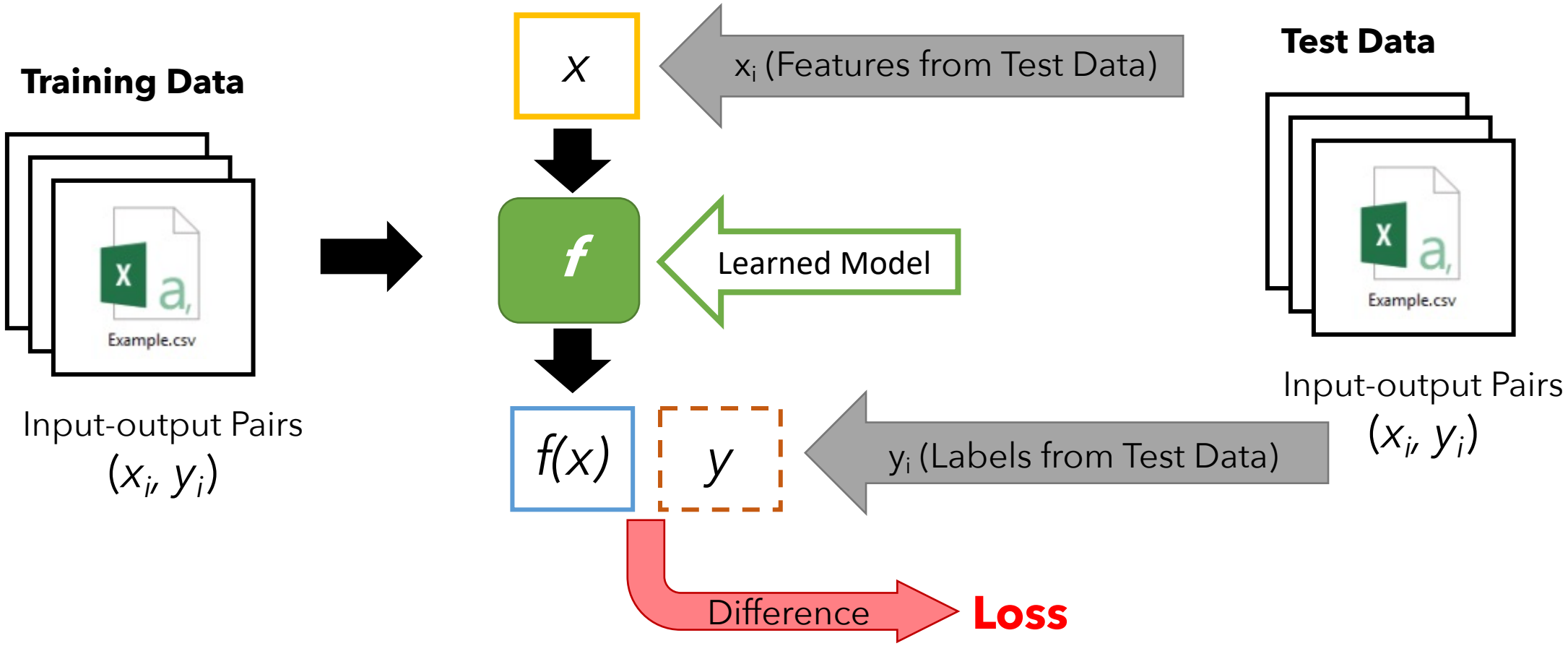
Previous Agenda

How do we evaluate model performance?

Today's Agenda

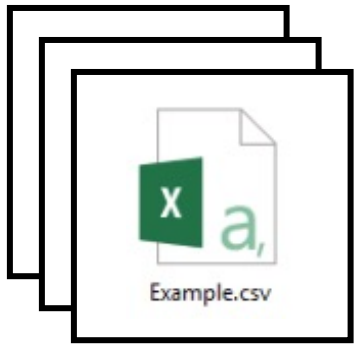
How do we implement models?
How do we evaluate our implementations?

Recall: Learning and Loss



Pandas

Training Data



Input-output Pairs
(x_i, y_i)

Test Data

Input-output Pairs
(x_i, y_i)

Managing Data is Hard.

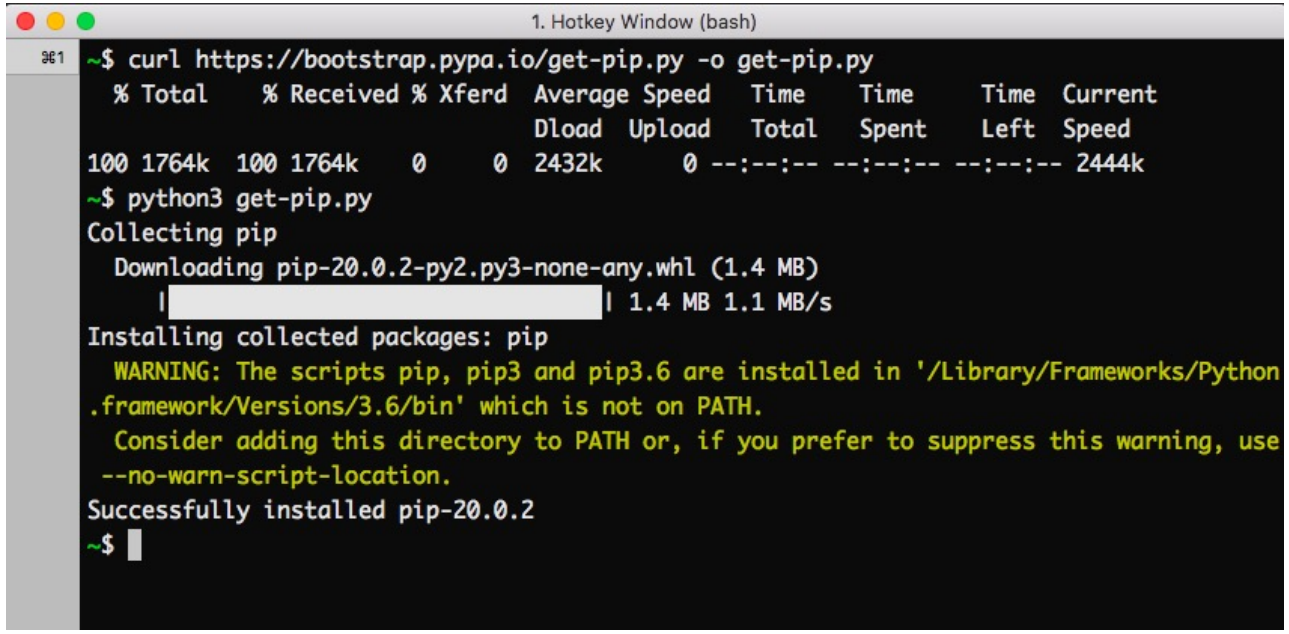
→ Excel? Interaction challenges.
→ Plain text? Even worse.

Pandas

→ A Python library for managing delimited data.

Name: "panel data" → pandas
Complementary to Other Libs:
→ matplotlib, scikit-learn

Python Libraries



Pip

→ Package manager for Python

→ Included by default for Python 3.4

https://docs.python.org/3/installing/index.html

Downloadable via Curl

→ Installs to your global env.

Usage Example: pandas

→ "pip install pandas"

Pandas

pandas_main.py

```
1   #   filename:   pandas.py
2   #   author:     alex williams
3   #   description:
4   #       this file provides a brief overview of the pandas library for managing data
5   import pandas as pd
6
7   # create a dict with k,v pairs that map fruits to lists of numbers
8   data = {
9       'apples': [3, 2, 0, 1],
10      'oranges': [0, 3, 7, 2]
11  }
12
13  # create a Pandas dataframe with the dict
14  purchases = pd.DataFrame(data)
15  print(purchases)
16
```

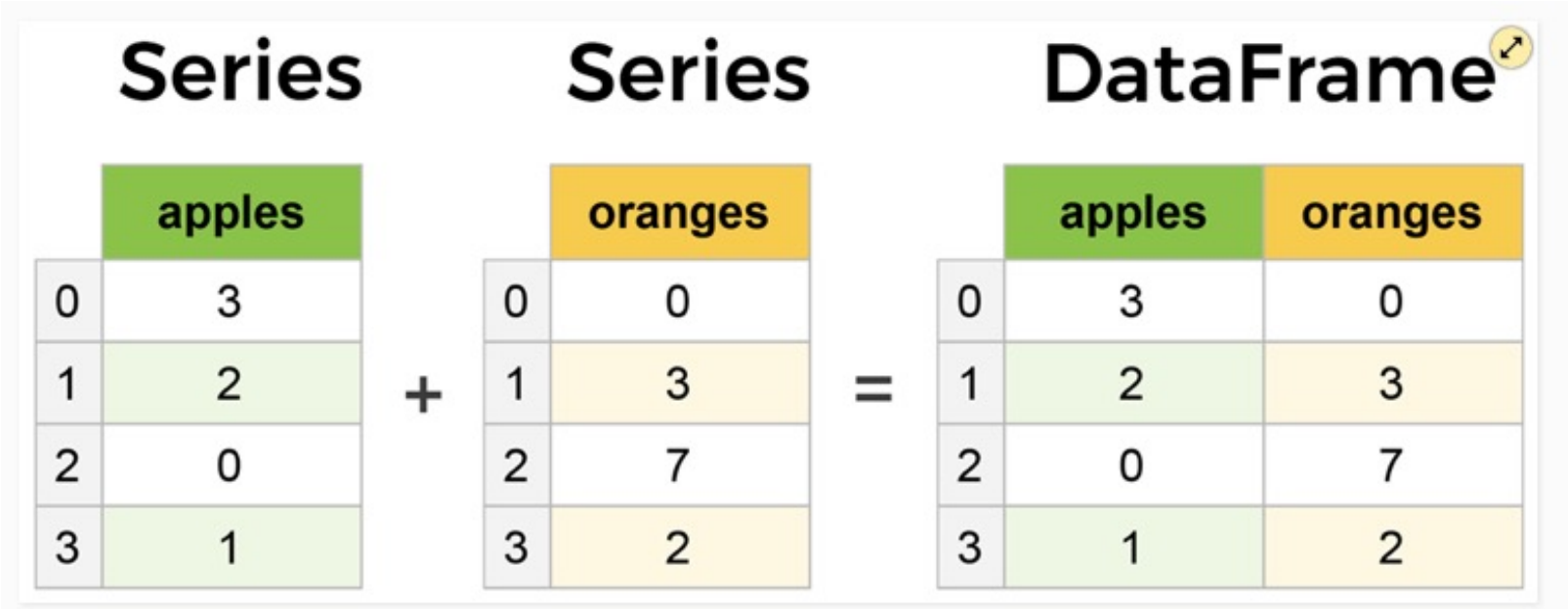
Dataframes

→ Primary data structure for Pandas

→ Optimized for speed / efficiency

```
ents/Project3$ python pandas_main.py
   apples  oranges
0       3        0
1       2        3
2       0        7
3       1        2
```

Pandas: Series



	Series		Series		DataFrame	
	apples		oranges		apples	oranges
0	3	+	0	=	3	0
1	2		3		2	3
2	0		7		0	7
3	1		2		1	2

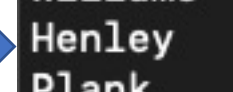
Understanding Dataframes

→ "Series" = Columns

→ Dataframe = Multi-dimensional table made up of Series

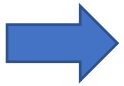
Pandas: Index

```
17  # create the dataframe with pre-defined indicies
18  purchases = pd.DataFrame(data, index=['Willams', 'Henley', 'Plank', 'Mockus'])
19  print(purchases)
20
```



```
          apples  oranges
Willams        3        0
Henley         2        3
Plank          0        7
Mockus         1        2
```

```
21  print(purchases.loc['Mockus'])
```



```
apples     1
oranges    2
Name: Mockus, dtype: int64
```

Dataframe Index

→ Allow us to perform look-ups on rows, e.g. with ".loc"

Pandas: Reading from CSVs

```
23  # alternatively, we could've had a CSV that included:
24  #   prof,apple,oranges
25  #   Williams,3,0
26  #   Henley,2,3
27  #   Plank,0,7
28  #   Mockus,1,2
29  #
30  #   and, read it in like:
31  purchases = pd.read_csv('purchases.csv', index_col=0)
32  print(purchases)
33
```

	apples	oranges
Willams	3	0
Henley	2	3
Plank	0	7
Mockus	1	2

Standard CSVs

→ Read in directly, specifying the first column as our index.

Pandas: Movie Dataset

```
34  # Example: IMDB Movies Dataset
35  movies_df = pd.read_csv("IMDB-Movies.csv", index_col="title")
36  print(movies_df.head())
```

title	id	imdb_id	popularity	...	release_year	budget_adj	revenue_adj
				...			
Jurassic World	135397	tt0369610	32.985763	...	2015	1.379999e+08	1.392446e+09
Mad Max: Fury Road	76341	tt1392190	28.419936	...	2015	1.379999e+08	3.481613e+08
Insurgent	262500	tt2908446	13.112507	...	2015	1.012000e+08	2.716190e+08
Star Wars: The Force Awakens	140607	tt2488496	11.173104	...	2015	1.839999e+08	1.902723e+09
Furious 7	168259	tt2820852	9.335014	...	2015	1.747999e+08	1.385749e+09

General APIs for Dataframes

→ head() = first 5 rows // head(10) = first 10 rows

→ tail() = last 5 rows

Pandas: Movie Dataset

Why Use Pandas?

→ It is designed to make managing data easy.

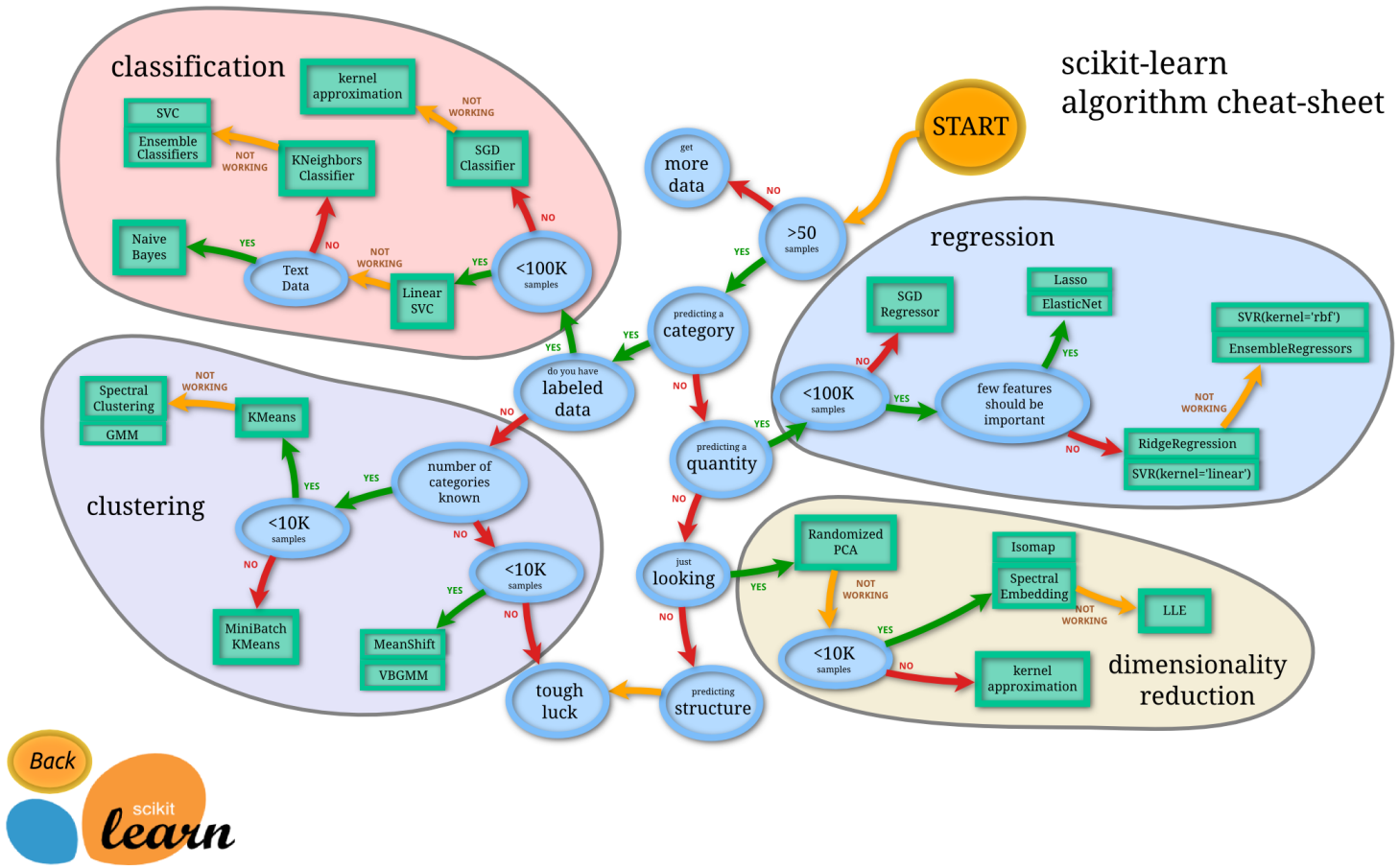
→ **Built-in functions for:**
- → Learning about your data
- → Expanding your data.

→ **Widespread support for Pandas**
- → The standard for most ML libraries.
- → Alternative Approach: Write everything yourself.

```
1  anime.groupby(["type"]).agg({
2    "rating": "sum",
3    "episodes": "count",
4    "name": "last"
5  }).reset_index()
```

	type	rating	episodes	name
0	Movie	14512.58	2348	Yasuji no Pornorama: Yacchimae!!
1	Music	2727.43	488	Yuu no Mahou
2	ONA	3679.43	659	Docchi mo Maid
3	OVA	20942.60	3311	Violence Gekiga Shin David no Hoshi: Inma Dens...
4	Special	10900.77	1676	Junjou Shoujo Et Cetera Specials
5	TV	25338.34	3787	Yuuki Yuuna wa Yuusha de Aru: Yuusha no Shou

Scikit-Learn



Scikit-Learn

→ Original library for ML

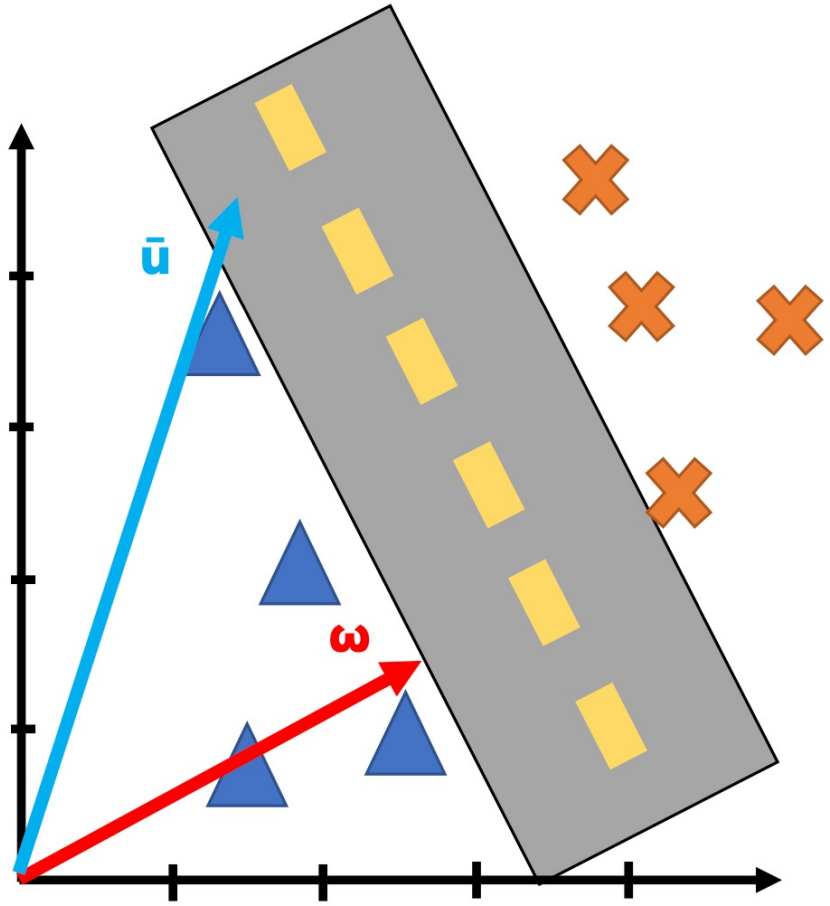
→ Includes most ML models

https://scikit-learn.org/

Downloadable via Pip

→ pip install scikit-learn

The "Widest Street" Analogy



The "best" linear separator is the one that produces the widest street.

ω: a vector that is perpendicular to the median line of the street.

ū: some unknown, that we have no positional awareness of.

At some point, the product is greater than some constant.

$$\omega \cdot \bar{u} > c$$

Support Vector Machines

→ Problem: Data not linearly separable.

→ **The SVM Approach**
"Create a new dimension"

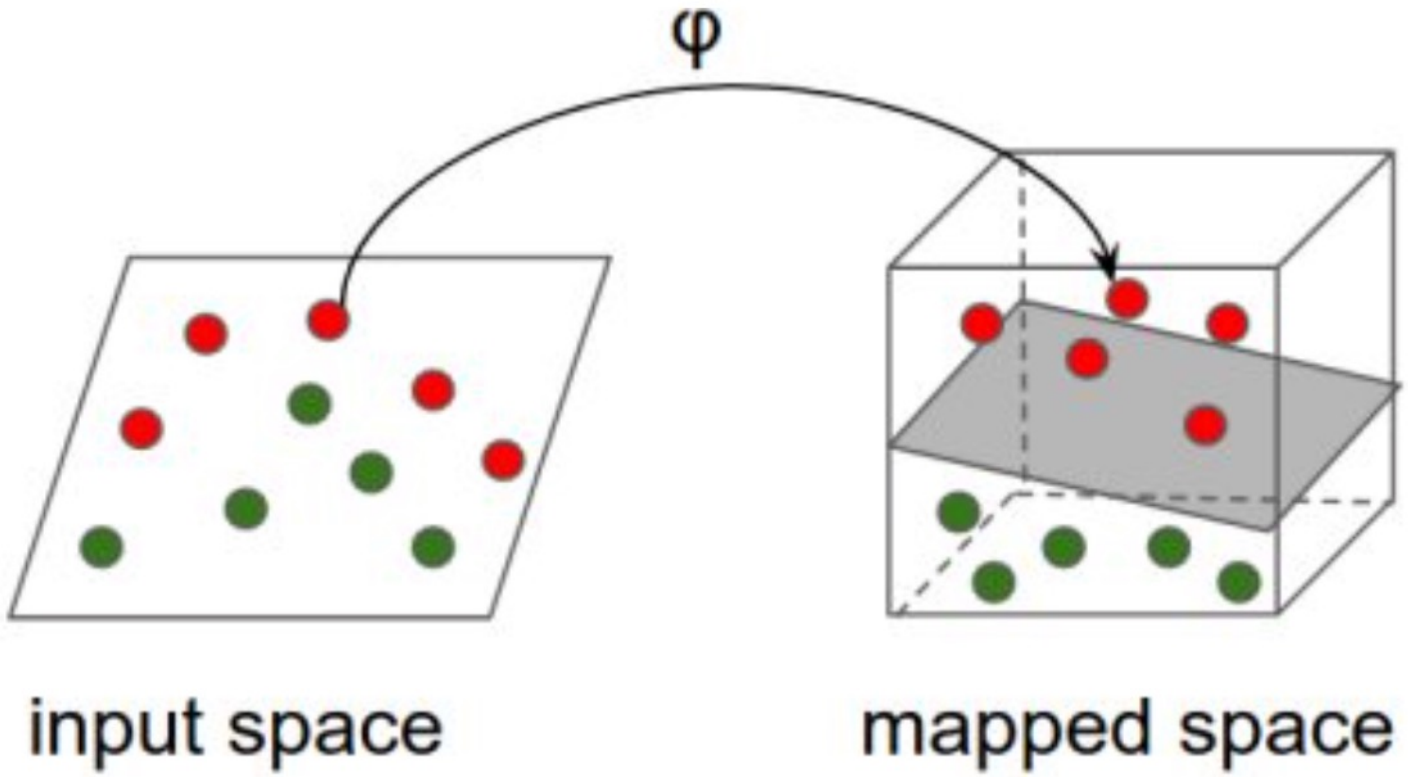
Hyperparameters:

1. Soft-margin Constant, C
→ *"How many points do we ignore near the margin?"*

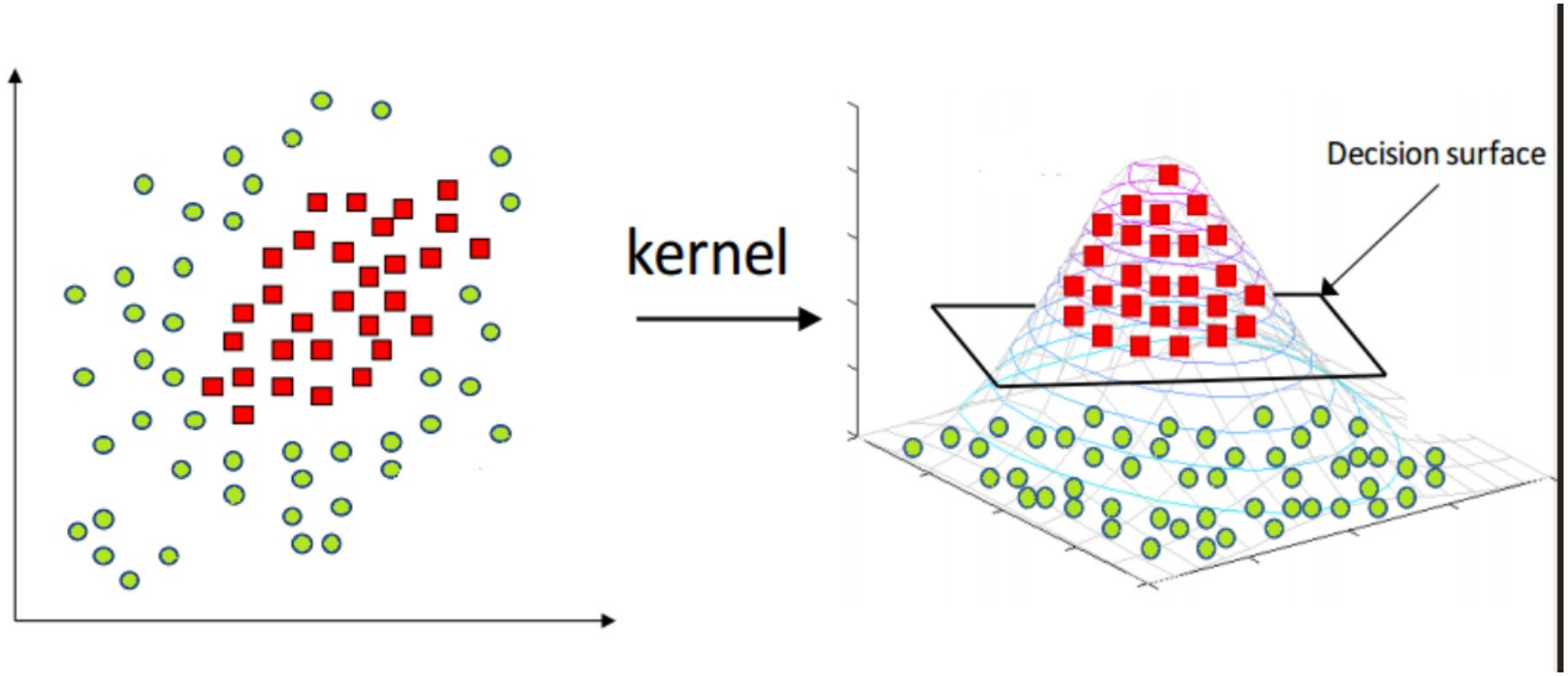
2. Kernel (e.g., Poly, RBF, Sigmoid)
→ *"What function can we apply to the data?"*

3. Degree (only for Poly)
→ *"How much should the boundary bend?"*

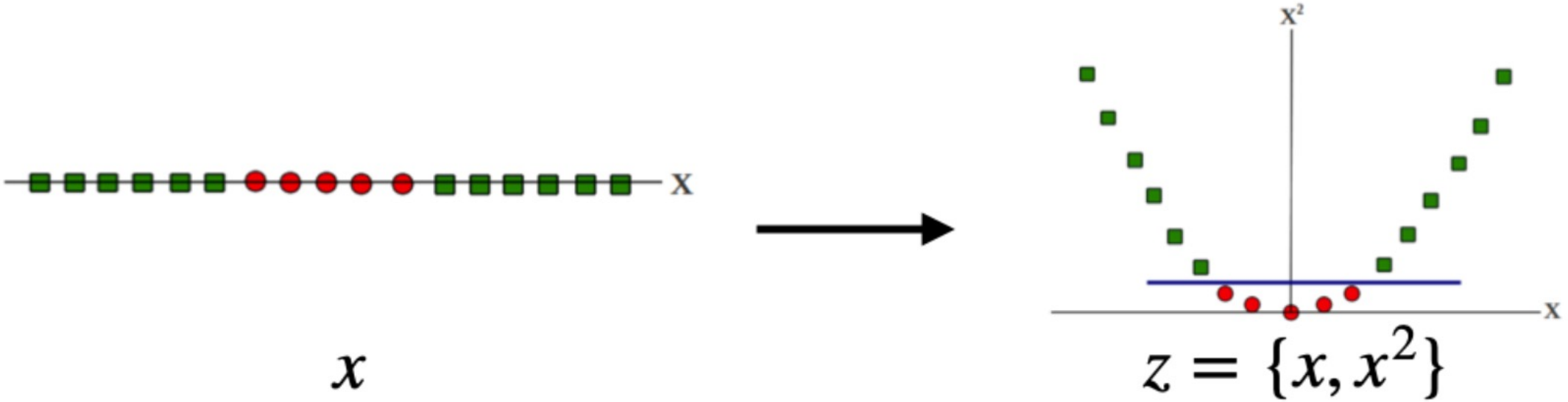
4. Gamma (only for RBF)
→ *"How tight do you want to fit to the data?"*



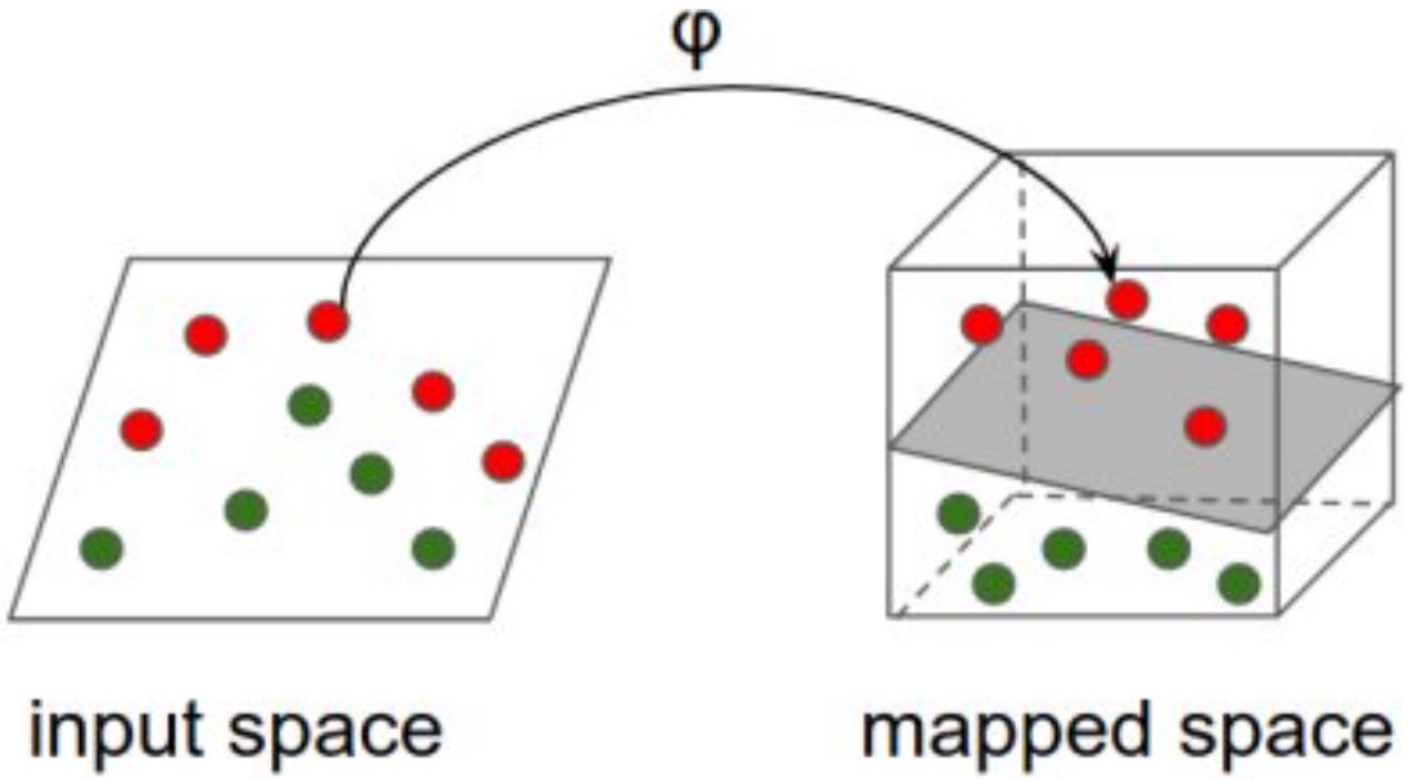
SVMs create a mapping that makes data linearly separable.



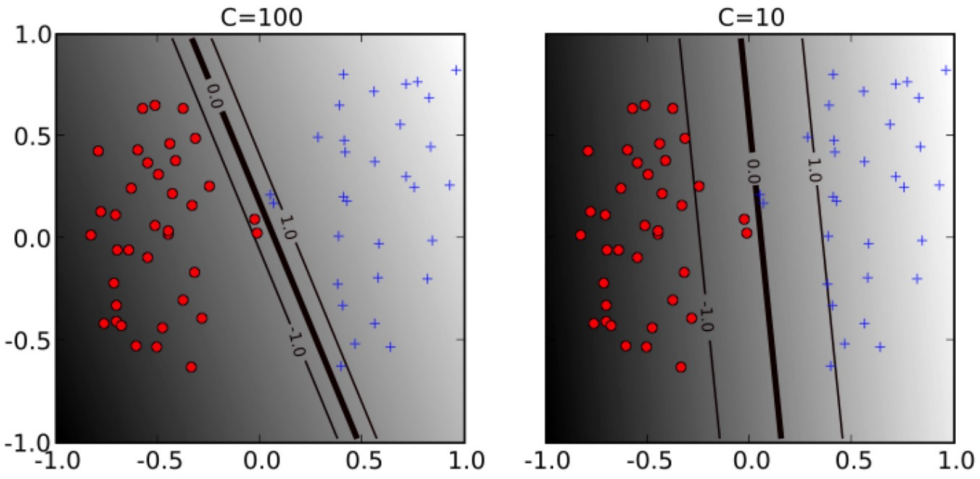
SVMs create a mapping that makes data linearly separable.



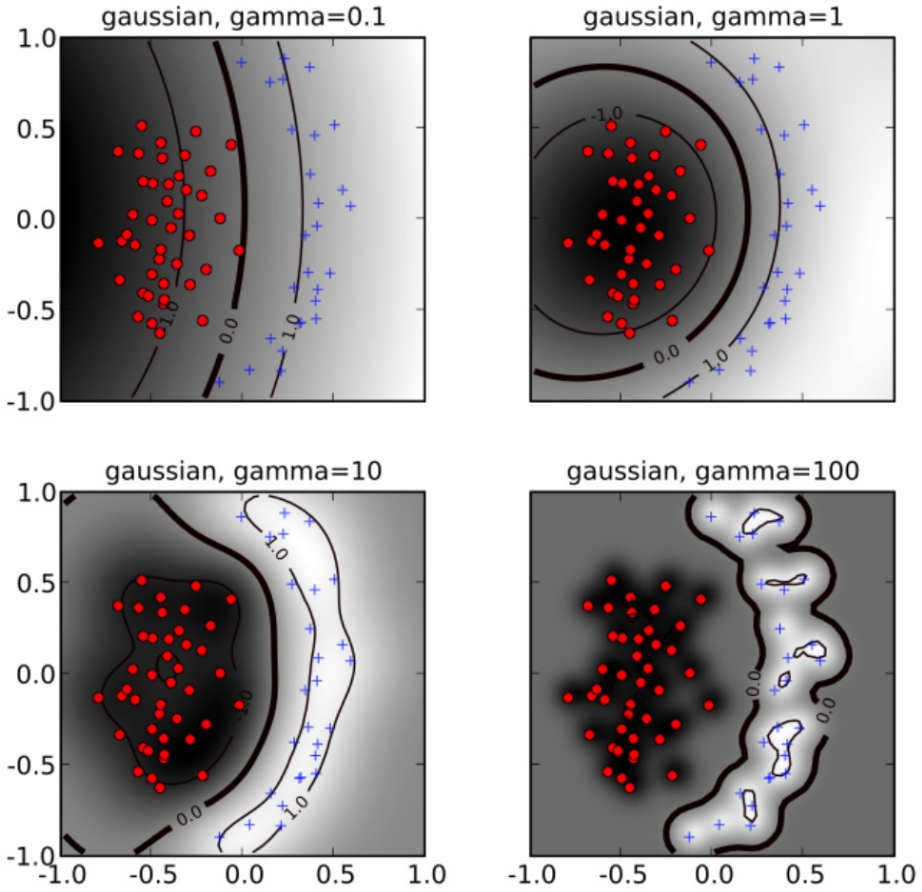
SVMs create a mapping that makes data linearly separable.



Pop Quiz: Do we always achieve separability?



C=100 vs C=10



G=0.1 vs G=1 vs G=10 vs G=100

SVMs with Scikit-learn

1.4. Support Vector Machines

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).

The support vector machines in scikit-learn support both dense (`numpy.ndarray` and convertible to that by `numpy.asarray`) and sparse (any `scipy.sparse`) sample vectors as input. However, to use an SVM to make predictions for sparse data, it must have been fit on such data. For optimal performance, use C-ordered `numpy.ndarray` (dense) or `scipy.sparse.csr_matrix` (sparse) with `dtype=float64`.

Scitkit-learn

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

Iris Data Set

Download: Data Folder, Data Set Description

Abstract: Famous database; from Fisher, 1936

Data Set Characteristics:	Multivariate	**Number of Instances:**	150	**Area:**	Life
Attribute Characteristics:	Real	**Number of Attributes:**	4	**Date Donated**	1988-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	3583041

Attribute Information:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
-- Iris Setosa
-- Iris Versicolour
-- Iris Virginica

SVMs with Scikit-learn

```
1   import numpy as np
2   import pandas as pd
3   import pylab as pl
4   from sklearn import svm, datasets
5   from sklearn.model_selection import train_test_split
6   from sklearn.model_selection import GridSearchCV
7   from sklearn.metrics import classification_report
8   from sklearn.metrics import confusion_matrix, accuracy_score
9   from sklearn.svm import SVC
```

Versions

- Python 3.7
- Matplotlib=3.1.2
- Numpy =1.17.2
- Pandas=0.25.1
- Scikit-learn=0.23.2

SVMs with Scikit-learn

```
10
11    # By default, Sklearn forces warnings into your terminal.
12    # Here, we're writing a dummy function that overwrites the function
13    # that prints out numerical warnings.
14    # (You probably don't want to do this in any of your projects!)
15    def warn(*args, **kwargs):
16        pass
17    import warnings
18    warnings.warn = warn
19
```

SVMs with Scikit-learn

Attribute Information:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
-- Iris Setosa
-- Iris Versicolour
-- Iris Virginica

```
20    # Load the IRIS dataset from Sklearn
21    # And convert it to a dataframe that we can manage with column names
22    iris = datasets.load_iris()
23    iris = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
24                        columns= iris['feature_names'] + ['target'])
25
26    # Define the two parameters we want to use.
27    features = ['petal length (cm)', 'petal width (cm)']
28    #features = ['sepal length (cm)', 'sepal width (cm)']
29
```

SVMs with Scikit-learn

```
30    # Separate our features from our targets.
31    X = iris[features]
32    Y = iris[['target']]
33
34    # Use Sklearn to get splits in our data for training and testing.
35    x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.8, random_state=0)
36    y_train_converted = y_train.values.ravel()
```

Scikit can handle test-train splits for you.

SVMs with Scikit-learn

```
38    # Now, we create several instances of SVCs that utilize varying kernels.
39    # We're not normalizing our data here because we want to plot the support vectors.
40    svc     = svm.SVC(kernel='linear').fit(x_train, y_train)
41    poly_svc_one = svm.SVC(kernel='poly', C=100, degree=1).fit(x_train, y_train_converted)
42    poly_svc_two = svm.SVC(kernel='poly', C=100, degree=2).fit(x_train, y_train_converted)
43    poly_svc_three = svm.SVC(kernel='poly', C=10, degree=3).fit(x_train, y_train_converted)
44    rbf_svc_one = svm.SVC(kernel='rbf', C=10, gamma=0.1).fit(x_train, y_train_converted)
45    rbf_svc_two = svm.SVC(kernel='rbf', C=100, gamma=0.1).fit(x_train, y_train_converted)
46    rbf_svc_three = svm.SVC(kernel='rbf', C=10, gamma=0.5).fit(x_train, y_train_converted)
47    rbf_svc_four = svm.SVC(kernel='rbf', C=100, gamma=0.5).fit(x_train, y_train_converted)
```

We generate eight models for the same dataset.
1 linear model, 3 SVM + Poly models, and 4 SVM + RBF models.

SVMs with Scikit-learn

```
49    # Now, we run our test data through our trained models.
50    predicted_linear = svc.predict(x_test)
51    predicted_poly_one = poly_svc_one.predict(x_test)
52    predicted_poly_two = poly_svc_two.predict(x_test)
53    predicted_poly_three = poly_svc_three.predict(x_test)
54    predicted_rbf_one = rbf_svc_one.predict(x_test)
55    predicted_rbf_two = rbf_svc_two.predict(x_test)
56    predicted_rbf_three = rbf_svc_three.predict(x_test)
57    predicted_rbf_four = rbf_svc_four.predict(x_test)
```

Model Prediction

We call the predict function on each model and capture the results.

SVMs with Scikit-learn

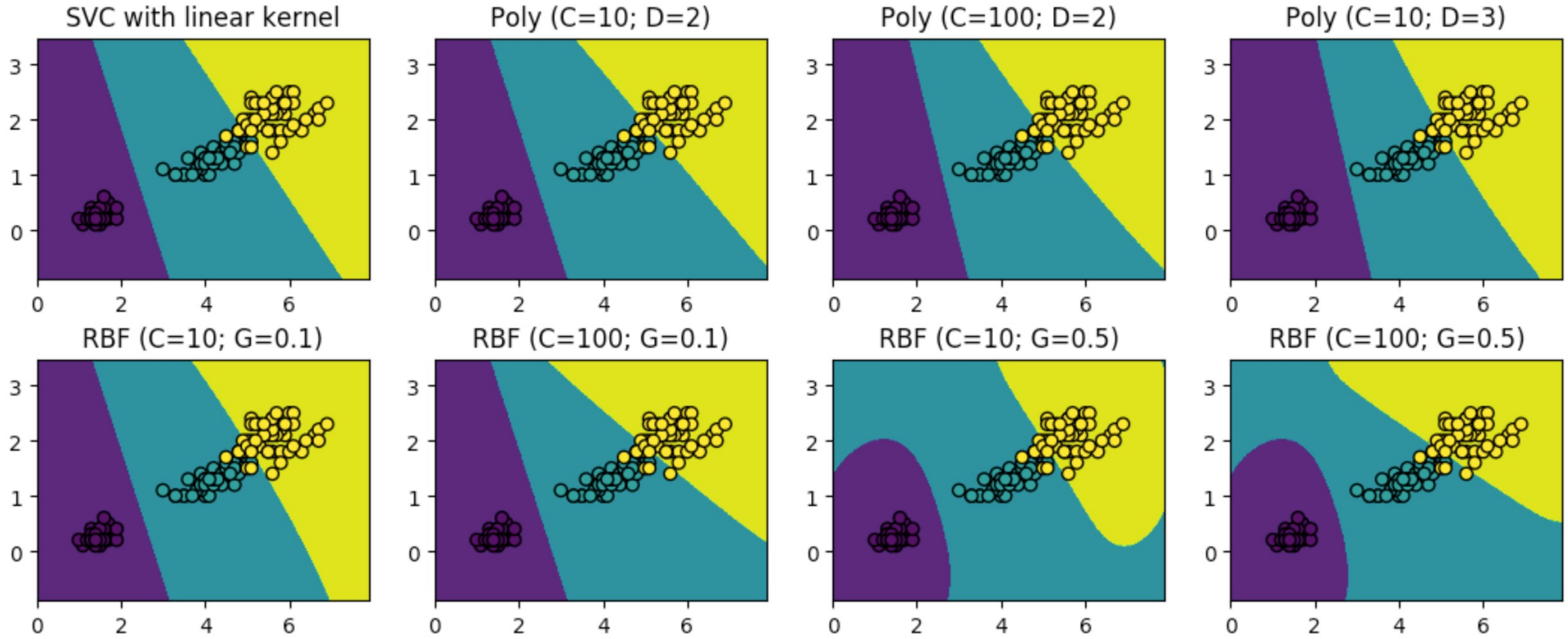
```
59    # Print our accuracies.
60    print("SVM + Linear \t\t-> " + str(accuracy_score(y_test, predicted_linear)))
61    print("SVM + Poly (D=1)\t-> " + str(accuracy_score(y_test, predicted_poly_one)))
62    print("SVM + Poly (D=2)\t-> " + str(accuracy_score(y_test, predicted_poly_two)))
63    print("SVM + Poly (D=3)\t-> " + str(accuracy_score(y_test, predicted_poly_three)))
64    print("SVM + RBF-10/0.1 \t-> " + str(accuracy_score(y_test, predicted_rbf_one)))
65    print("SVM + RBF-100/0.1 \t-> " + str(accuracy_score(y_test, predicted_rbf_two)))
66    print("SVM + RBF-10/0.5 \t-> " + str(accuracy_score(y_test, predicted_rbf_three)))
67    print("SVM + RBF-100/0.5 \t-> " + str(accuracy_score(y_test, predicted_rbf_four)))
```

```
(base) alex@MacBook-Pro:~/Desktop/COSC425/SVM-Example$ python svm.py
SVM + Linear            -> 0.9416666666666667
SVM + Poly (D=1)        -> 0.9083333333333333
SVM + Poly (D=2)        -> 0.9416666666666667
SVM + Poly (D=3)        -> 0.9333333333333333
SVM + RBF-10/0.1        -> 0.9166666666666666
SVM + RBF-100/0.1       -> 0.9166666666666666
SVM + RBF-10/0.5        -> 0.9
SVM + RBF-100/0.5       -> 0.9333333333333333
```

Model Accuracy

We call sklearn's accuracy_score function to compute the difference between predicted and observed target labels.

SVMs with Scikit-learn



Visualizing Boundaries

Source code included in the svm.py file on Canvas.

SVMs with Scikit-learn

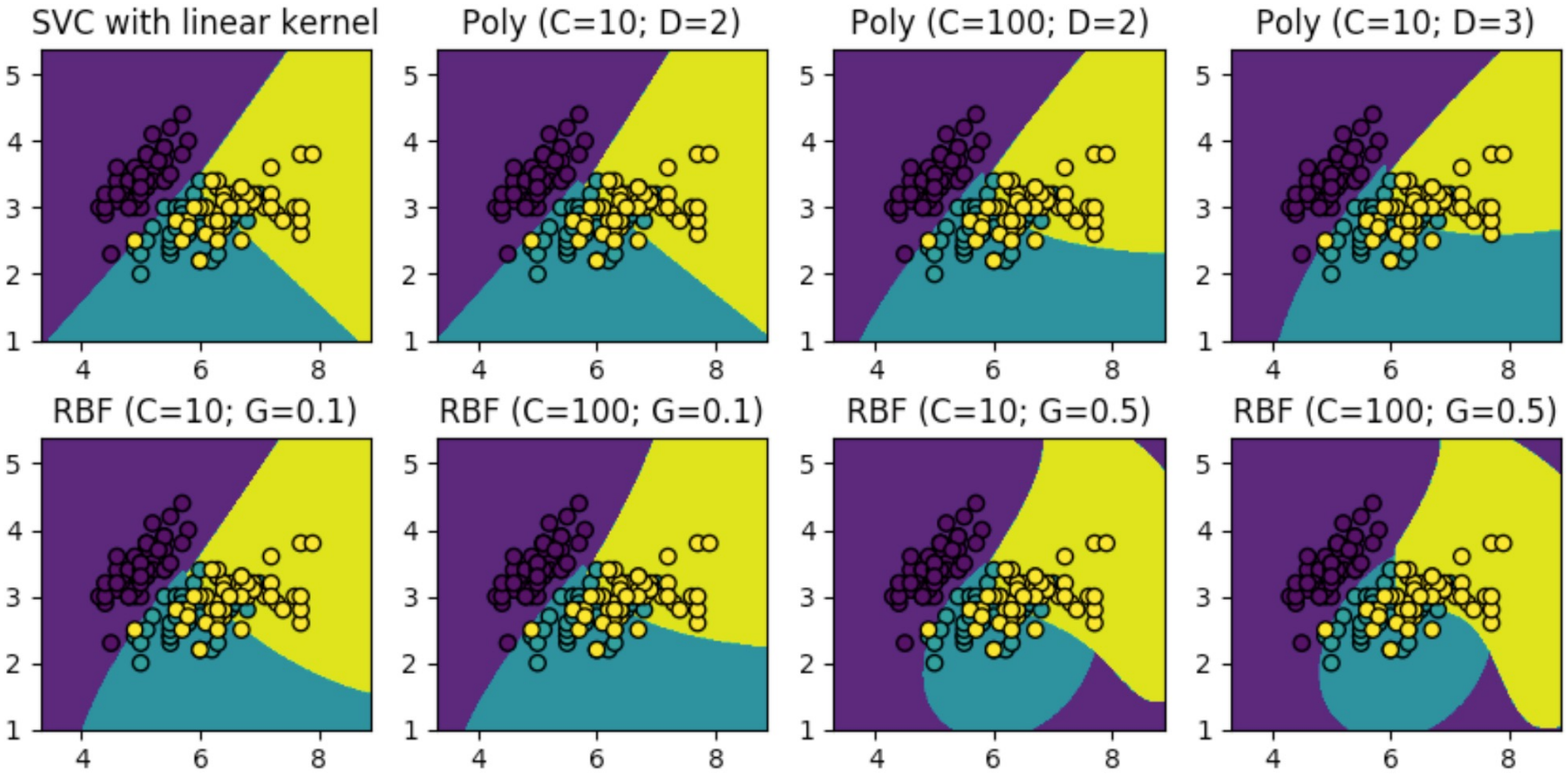
```
20   # Load the IRIS dataset from Sklearn
21   # And convert it to a dataframe that we can manage with column names.
22   iris = datasets.load_iris()
23   iris = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
24                       columns= iris['feature_names'] + ['target'])
25
26   # Define the two parameters we want to use.
27   #features = ['petal length (cm)', 'petal width (cm)']
28   features = ['sepal length (cm)', 'sepal width (cm)']
```

Changed features.

Let's Look at Sepal Length and Width.

Petal length and width were *very* linearly separable.

SVMs with Scikit-learn



Visualizing Boundaries

Using these features, our data is *far* less linearly separable than before.

SVMs with Scikit-learn

```
123  # # # # # # # # # # # # # # # # # # # # #
124  # Coarse Grid Search                      #
125  #   - Broad sweep of hyperparemeters.     #
126  # # # # # # # # # # # # # # # # # # # # #
127
128  # Set the parameters by cross-validation
129  tuned_parameters = [
130      {
131          'kernel': ['linear'],
132          'C': [1, 10, 100, 1000]
133      },
134      {
135          'kernel': ['poly'],
136          'degree': [2, 3, 4],
137          'C': [1, 10, 100, 1000]
138      },
139      {
140          'kernel': ['rbf'],
141          'gamma': [1e-3, 1e-4],
142          'C': [1, 10, 100, 1000]
143      }
144  ]
```

Coarse Grid Search

A broad sweep of hyperparameters.

SVMs with Scikit-learn

```
123  # # # # # # # # # # # # # # # # # # # #
124  # Coarse Grid Search                    #
125  #   - Broad sweep of hyperparemeters.   #
126  # # # # # # # # # # # # # # # # # # # #
127
128  # Set the parameters by cross-validation
129  tuned_parameters = [
130      {
131          'kernel': ['linear'],
132          'C': [1, 10, 100, 1000]
133      },
134      {
135          'kernel': ['poly'],
136          'degree': [2, 3, 4],
137          'C': [1, 10, 100, 1000]
138      },
139      {
140          'kernel': ['rbf'],
141          'gamma': [1e-3, 1e-4],
142          'C': [1, 10, 100, 1000]
143      }
144  ]
```

```
147  scores = ['precision', 'recall']
148
149  for score in scores:
150      print("# Tuning hyper-parameters for %s" % score)
151      print()
152
153      clf = GridSearchCV(
154          SVC(), tuned_parameters, scoring='%s_macro' % score
155      )
156      clf.fit(x_train, y_train)
```

Grid Search
The GridSearchCV function exhaustively explores combinations of parameters to determine the "best" performing set.

Scores = Which metrics do you want to tune your parameters toward?

SVMs with Scikit-learn

```
print("Best parameters set found on development set:")
print()
print(clf.best_params_)
print()
print("Grid scores on development set:")
print()
means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, clf.cv_results_['params']):
    print("%0.3f (+/-%0.03f) for %r"
          % (mean, std * 2, params))
print()
```

```
# Tuning hyper-parameters for precision

Best parameters set found on development set:

{'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}

Grid scores on development set:

0.978 (+/-0.089) for {'C': 1, 'kernel': 'linear'}
0.978 (+/-0.089) for {'C': 10, 'kernel': 'linear'}
0.978 (+/-0.089) for {'C': 100, 'kernel': 'linear'}
0.978 (+/-0.089) for {'C': 1000, 'kernel': 'linear'}
0.978 (+/-0.089) for {'C': 1, 'degree': 2, 'kernel': 'poly'}
0.978 (+/-0.089) for {'C': 1, 'degree': 3, 'kernel': 'poly'}
0.978 (+/-0.089) for {'C': 1, 'degree': 4, 'kernel': 'poly'}
0.978 (+/-0.089) for {'C': 10, 'degree': 2, 'kernel': 'poly'}
0.978 (+/-0.089) for {'C': 10, 'degree': 3, 'kernel': 'poly'}
0.978 (+/-0.089) for {'C': 10, 'degree': 4, 'kernel': 'poly'}
0.978 (+/-0.089) for {'C': 100, 'degree': 2, 'kernel': 'poly'}
0.978 (+/-0.089) for {'C': 100, 'degree': 3, 'kernel': 'poly'}
0.978 (+/-0.089) for {'C': 100, 'degree': 4, 'kernel': 'poly'}
0.978 (+/-0.089) for {'C': 1000, 'degree': 2, 'kernel': 'poly'}
0.978 (+/-0.089) for {'C': 1000, 'degree': 3, 'kernel': 'poly'}
0.978 (+/-0.089) for {'C': 1000, 'degree': 4, 'kernel': 'poly'}
0.289 (+/-0.711) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.289 (+/-0.711) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.600 (+/-0.400) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.289 (+/-0.711) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
1.000 (+/-0.000) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.600 (+/-0.400) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.978 (+/-0.089) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
1.000 (+/-0.000) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
```

Tuning for Precision

We see RBF w/ C=100, G=0.001 as the best performing model.

SVMs with Scikit-learn

```
print("Best parameters set found on development set:")
print()
print(clf.best_params_)
print()
print("Grid scores on development set:")
print()
means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, clf.cv_results_['params']):
    print("%0.3f (+/-%0.03f) for %r"
          % (mean, std * 2, params))
print()
```

```
# Tuning hyper-parameters for recall

Best parameters set found on development set:

{'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}

Grid scores on development set:

0.967 (+/-0.133) for {'C': 1, 'kernel': 'linear'}
0.967 (+/-0.133) for {'C': 10, 'kernel': 'linear'}
0.967 (+/-0.133) for {'C': 100, 'kernel': 'linear'}
0.967 (+/-0.133) for {'C': 1000, 'kernel': 'linear'}
0.967 (+/-0.133) for {'C': 1, 'degree': 2, 'kernel': 'poly'}
0.967 (+/-0.133) for {'C': 1, 'degree': 3, 'kernel': 'poly'}
0.967 (+/-0.133) for {'C': 1, 'degree': 4, 'kernel': 'poly'}
0.967 (+/-0.133) for {'C': 10, 'degree': 2, 'kernel': 'poly'}
0.967 (+/-0.133) for {'C': 10, 'degree': 3, 'kernel': 'poly'}
0.967 (+/-0.133) for {'C': 10, 'degree': 4, 'kernel': 'poly'}
0.967 (+/-0.133) for {'C': 100, 'degree': 2, 'kernel': 'poly'}
0.967 (+/-0.133) for {'C': 100, 'degree': 3, 'kernel': 'poly'}
0.967 (+/-0.133) for {'C': 100, 'degree': 4, 'kernel': 'poly'}
0.967 (+/-0.133) for {'C': 1000, 'degree': 2, 'kernel': 'poly'}
0.967 (+/-0.133) for {'C': 1000, 'degree': 3, 'kernel': 'poly'}
0.967 (+/-0.133) for {'C': 1000, 'degree': 4, 'kernel': 'poly'}
0.467 (+/-0.533) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.467 (+/-0.533) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.733 (+/-0.267) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.467 (+/-0.533) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
1.000 (+/-0.000) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.733 (+/-0.267) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.967 (+/-0.133) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
1.000 (+/-0.000) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
```

Tuning for Recall

Again, we see RBF w/ C=100, G=0.001 as a winner.

SVMs with Scikit-learn

```
171     print("Detailed classification report:")
172     print()
173     print("The model is trained on the full development set.")
174     print("The scores are computed on the full evaluation set.")
175     print()
176     y_true, y_pred = y_test, clf.predict(x_test)
177     print(classification_report(y_true, y_pred))
178     print()
```

```
Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        40
         1.0       0.88      0.97      0.93        39
         2.0       0.97      0.88      0.92        41

    accuracy                           0.95       120
   macro avg       0.95      0.95      0.95       120
weighted avg       0.95      0.95      0.95       120
```

Classification Report
Summarized across all target classes.
→ Useful for high-level perspective.

SVMs with Scikit-learn

```
# # # # # # # # # # # # # # # # # # # # #
# Coarse Grid Search                      #
#   - Broad sweep of hyperparemeters.     #
# # # # # # # # # # # # # # # # # # # # #

# Set the parameters by cross-validation
tuned_parameters = [
    {
        'kernel': ['linear'],
        'C': [1, 10, 100, 1000]
    },
    {
        'kernel': ['poly'],
        'degree': [2, 3, 4],
        'C': [1, 10, 100, 1000]
    },
    {
        'kernel': ['rbf'],
        'gamma': [1e-3, 1e-4],
        'C': [1, 10, 100, 1000]
    }
]
```

We want to refine the parameters we're seeking to tune.

Fine Grid Search
A grid search that follows the coarse search with more detailed parameters.

We saw that RBF w/ C=100, G=0.001 was our best performing model.
→ We can commit to using RBF.
→ We want to dive deeper with smaller tweaks for C and G.

Stopping Point: We've examined a wealth of fine-grained possibilities and feel confident that we've explored enough.

Scikit-Learn: Neural Nets

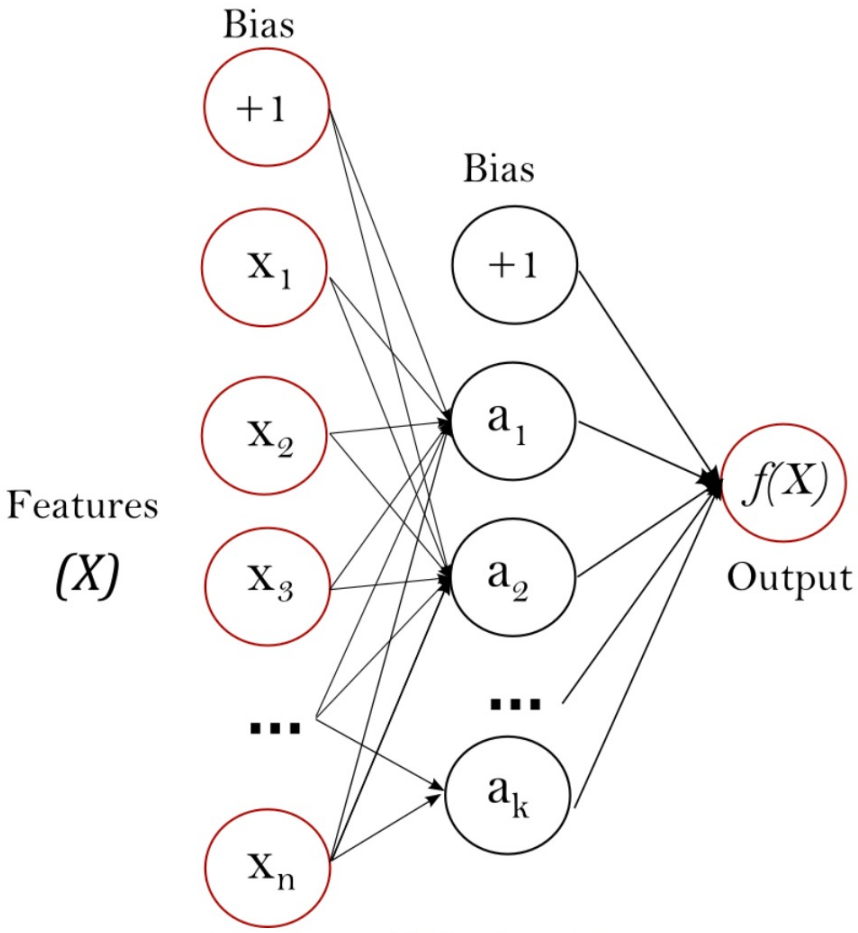


Figure 1 : One hidden layer MLP.

I. NN Training

```
>>> from sklearn.neural_network import MLPClassifier
>>> X = [[0., 0.], [1., 1.]]
>>> y = [0, 1]
>>> clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
...                     hidden_layer_sizes=(5, 2), random_state=1)
...
>>> clf.fit(X, y)
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(5, 2), random_state=1,
              solver='lbfgs')
```

II. NN Usage

```
>>> clf.predict([[2., 2.], [-1., -2.]])
array([1, 0])
```

III. Examining Weights

```
>>> [coef.shape for coef in clf.coefs_]
[(2, 5), (5, 2), (2, 1)]
```

Critiquing Scikit-Learn

```
>>> from sklearn.neural_network import MLPClassifier
>>> X = [[0., 0.], [1., 1.]]
>>> y = [0, 1]
>>> clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
...                     hidden_layer_sizes=(5, 2), random_state=1)
...
>>> clf.fit(X, y)
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(5, 2), random_state=1,
              solver='lbfgs')
```

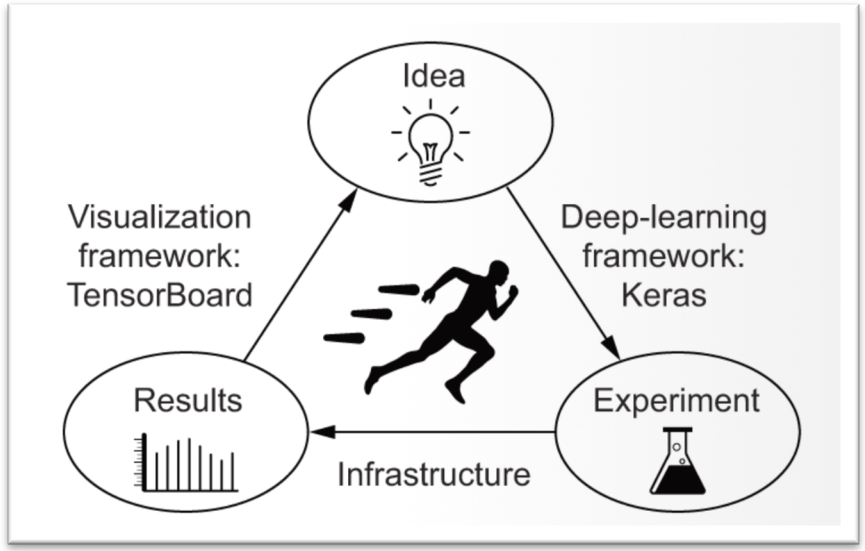
Considerations for Implementation

→ Feasibility, i.e., Is it possible implement what we want?
→ Usability, i.e., Is it easy to implement what we want?
→ Efficiency, i.e., Is it quick to implement what we want?

Scikit-learn has near-unlimited feasibility with severely limited usability.

→ Code is often unintuitive and often unreadable.
→ You can implement most, if not all, standard ML approaches.

Tensorflow and Keras



Tensorflow (https://www.tensorflow.org/)

→ Mainstream ML library supported *just about* everywhere.

→ Neural Networks with a more mathematical emphasis.

→ Requires a technical background to use.

Keras (https://keras.io)

→ A Deep Learning library that operates on top of other machine learning libraries.

→ Designed to be "scalable" and "user-friendly".

→ Provides a high-level API for building neural nets.

→ OOP for Neural Nets

NNs with Keras and Tensorflow

```
1   from sklearn import datasets
2   import numpy as np
3   import pandas as pd
4   from sklearn.model_selection import train_test_split
5   from sklearn.preprocessing import MinMaxScaler
6   from keras.models import Sequential
7   from keras.layers import Dense
8   from sklearn.preprocessing import LabelBinarizer
9   from keras.wrappers.scikit_learn import KerasClassifier
10  from sklearn.model_selection import KFold
11  from sklearn.model_selection import cross_val_score
```

Versions

- Python 3.7
- Matplotlib=3.1.2
- Numpy =1.17.2
- Pandas=0.25.1
- Scikit-learn=0.23.2
- Keras=2.4.3

nn.py on Canvas

NNs with Keras and Tensorflow

```
13  # (0) Hide as many warnings as possible!
14  import os
15  import tensorflow as tf
16  os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
17  tf.get_logger().setLevel('INFO')
18  tf.compat.v1.disable_eager_execution()
19
20  # (1) Read in the Iris dataset.
21  iris = datasets.load_iris()
22  iris_data_df = pd.DataFrame(data=iris.data, columns=iris.feature_names,
23                              dtype=np.float32)
```

Varies between versions
→ Be careful!

Iris Dataframe
→ Three target classes.

Hiding Tensorflow Warnings + Read-in Data

NNs with Keras and Tensorflow

```
25    # (2) Create an encoder that "binarizes" target labels.
26    # e.g. We have 3 target classes. When we instantiate and use fit_transform() on an
27    #      encoder, the function returns a N x 3 dataframe. Each row in this new dataframe
28    #      will be equal to 0 or 1 based on whether the target class is true.
29    encoder = LabelBinarizer()
30    target = encoder.fit_transform(iris.target)
31    iris_target_df = pd.DataFrame(data=target, columns=iris.target_names)
```

The Keras API: Encoders

→ When applied, LabelBinarizer produces [Val1, Val2, Val3]

Each Val is equal to 0 or 1 based on an example's associated target label.

NNs with Keras and Tensorflow

```
33  # (3) Perform test-train splits.
34  X_train,X_test,y_train,y_test = train_test_split(iris_data_df,
35                                                   iris_target_df,
36                                                   test_size=0.30)
37
38  # (4) Perform standardization on our data.
39  scaler = MinMaxScaler(feature_range=(0,1))
40  X_train = pd.DataFrame(scaler.fit_transform(X_train),
41                                                   columns=X_train.columns,
42                                                   index=X_train.index)
43  X_test = pd.DataFrame(scaler.transform(X_test),
44                                                 columns=X_test.columns,
45                                                 index=X_test.index)
46
```

Perform Test-Train splits and Standardization.

→ Note the test_size variable in test_train_split().

NNs with Keras and Tensorflow

```
48  # (5) Build Keras models.
49  # # # # # # # # # # # # # # # # # # #
50  #   Model 1:   A Baseline Model #
51  # # # # # # # # # # # # # # # # # # #
52  def BaselineModel():
53      """ A sequential Keras model that has an input layer, one
54          hidden layer, and an output layer."""
55      model = Sequential()
56      model.add(Dense(4, input_dim=4, activation='sigmoid', name='layer_1'))
57      model.add(Dense(3, activation='sigmoid', name='output_layer'))
58
59      # Don't change this!
60      model.compile(loss="categorical_crossentropy",
61                    optimizer="adam",
62                    metrics=['accuracy'])
63      return model
```

Building a Baseline Model

→ Here, we're using the Sequential API to build a baseline model.

NNs with Keras and Tensorflow

```
66  # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
67  #   Model 2:   A Model with a Second Hidden Layer (sigmoid) #
68  # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
69  def AlternativeModel1():
70      """ A sequential Keras model that has an input layer, two
71          hidden layers, and an output layer."""
72      model = Sequential()
73      model.add(Dense(4, input_dim=4, activation='sigmoid', name='layer_1'))
74      model.add(Dense(4, activation='sigmoid', name='layer_2'))
75      model.add(Dense(3, activation='sigmoid', name='output_layer'))
76
77      # Don't change this!
78      model.compile(loss="categorical_crossentropy",
79                    optimizer="adam",
80                    metrics=['accuracy'])
81      return model
82
```

New Layer + Activation: “sigmoid”

```
83  # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
84  #   Model 3:   A Model with a Second Hidden Layer (tanh)  #
85  # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
86  def AlternativeModel2():
87      """ A sequential Keras model that has an input layer, two
88          hidden layers, and an output layer."""
89      model = Sequential()
90      model.add(Dense(4, input_dim=4, activation='sigmoid', name='layer_1'))
91      model.add(Dense(4, activation='tanh', name='layer_2'))
92      # model.add(Dense(10, activation='tanh', name='layer_3'))
93      # model.add(Dense(10, activation='tanh', name='layer_4'))
94      model.add(Dense(3, activation='sigmoid', name='output_layer'))
95
96      # Don't change this!
97      model.compile(loss="categorical_crossentropy",
98                    optimizer="adam",
99                    metrics=['accuracy'])
100     return model
101
```

New Layer + Activation: “tanh”

Building Alternative Models

→ Models with an additional layer + different activation function.

NNs with Keras and Tensorflow

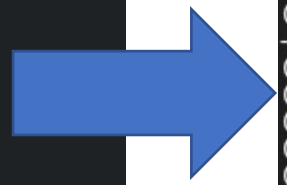
```
102   # (6) Model Evaluations
103   # Below, we build KerasClassifiers using our model definitions. Use verbose=2 to see
104   # real-time updates for each epoch.
105
106   # - - Model 1 - -
107   estimator = KerasClassifier(
108           build_fn=BaselineModel,
109           epochs=200, batch_size=20,
110           verbose=0)
111   kfold = KFold(n_splits=5, shuffle=True)
112   print("- - - - - - - - - - - - - - ")
113   for i in range(0,10):
114       results = cross_val_score(estimator, X_train, y_train, cv=kfold)
115       print("(MODEL 1 : RUN " + str(i) +") Performance: mean: %.2f%% std: (%.2f%%)" % (results.mean()*100, results.std()*100))
```

Model Evaluation

→ Pass models to KerasClassifiers, run epochs, and perform KFold.

NNs with Keras and Tensorflow

```
106   # - - Model 1 - -
107   estimator = KerasClassifier(
108           build_fn=BaselineModel,
109           epochs=200, batch_size=20,
110           verbose=0)
111   kfold = KFold(n_splits=5, shuffle=True)
112   print("- - - - - - - - - - - - - ")
113   for i in range(0,10):
114       results = cross_val_score(estimator, X_train, y_train, cv=kfold)
115       print("(MODEL 1 : RUN " + str(i) +") Performance: mean: %.2f%% std:
116
117   # - - Model 2 - -
118   estimator = KerasClassifier(
119           build_fn=AlternativeModel1,
120           epochs=200, batch_size=20,
121           verbose=0)
122   kfold = KFold(n_splits=5, shuffle=True)
123   print("- - - - - - - - - - - - - ")
124   for i in range(0,10):
125       results = cross_val_score(estimator, X_train, y_train, cv=kfold)
126       print("(MODEL 2 : RUN " + str(i) +") Performance: mean: %.2f%% std:
127
128   # - - Model 3 - -
129   estimator = KerasClassifier(
130           build_fn=AlternativeModel2,
131           epochs=200, batch_size=20,
132           verbose=0)
133   kfold = KFold(n_splits=5, shuffle=True)
134   print("- - - - - - - - - - - - - ")
135   for i in range(0,10):
136       results = cross_val_score(estimator, X_train, y_train, cv=kfold)
137       print("(MODEL 3 : RUN " + str(i) +") Performance: mean: %.2f%% std:
138
```



```
(base) alex@MacBook-Pro:~/Desktop/COSC425/NN-Example$ python nn.py
- - - - - - - - - - - - -
WARNING:tensorflow:From /Users/alex/opt/anaconda3/lib/python3.7/sit
is deprecated and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are
(MODEL 1 : RUN 0) Performance: mean: 50.48% std: (21.21%)
(MODEL 1 : RUN 1) Performance: mean: 55.24% std: (20.78%)
(MODEL 1 : RUN 2) Performance: mean: 64.76% std: (20.56%)
(MODEL 1 : RUN 3) Performance: mean: 56.19% std: (20.29%)
(MODEL 1 : RUN 4) Performance: mean: 46.67% std: (10.17%)
(MODEL 1 : RUN 5) Performance: mean: 67.62% std: (11.02%)
(MODEL 1 : RUN 6) Performance: mean: 82.86% std: (14.00%)
(MODEL 1 : RUN 7) Performance: mean: 52.38% std: (17.82%)
(MODEL 1 : RUN 8) Performance: mean: 44.76% std: (16.66%)
(MODEL 1 : RUN 9) Performance: mean: 59.05% std: (12.27%)
- - - - - - - - - - - - -
(MODEL 2 : RUN 0) Performance: mean: 53.33% std: (16.33%)
(MODEL 2 : RUN 1) Performance: mean: 46.67% std: (22.42%)
(MODEL 2 : RUN 2) Performance: mean: 57.14% std: (28.41%)
(MODEL 2 : RUN 3) Performance: mean: 46.67% std: (12.20%)
(MODEL 2 : RUN 4) Performance: mean: 40.95% std: (16.93%)
(MODEL 2 : RUN 5) Performance: mean: 31.43% std: (20.11%)
(MODEL 2 : RUN 6) Performance: mean: 63.81% std: (3.81%)
(MODEL 2 : RUN 7) Performance: mean: 48.57% std: (16.88%)
(MODEL 2 : RUN 8) Performance: mean: 56.19% std: (20.95%)
(MODEL 2 : RUN 9) Performance: mean: 30.48% std: (16.93%)
- - - - - - - - - - - - -
(MODEL 3 : RUN 0) Performance: mean: 68.57% std: (14.94%)
(MODEL 3 : RUN 1) Performance: mean: 63.81% std: (11.11%)
(MODEL 3 : RUN 2) Performance: mean: 73.33% std: (14.63%)
(MODEL 3 : RUN 3) Performance: mean: 80.00% std: (10.17%)
(MODEL 3 : RUN 4) Performance: mean: 69.52% std: (22.05%)
(MODEL 3 : RUN 5) Performance: mean: 69.52% std: (17.20%)
(MODEL 3 : RUN 6) Performance: mean: 76.19% std: (14.13%)
(MODEL 3 : RUN 7) Performance: mean: 67.62% std: (15.18%)
(MODEL 3 : RUN 8) Performance: mean: 64.76% std: (15.24%)
(MODEL 3 : RUN 9) Performance: mean: 72.38% std: (13.93%)
```

How do we search model configurations more efficiently?

(**nn_tuning.py on Canvas**)

NNs with Keras and Tensorflow

```
55  # (5) Build Keras models.
56  # # # # # # # # # # # # # # # # # #
57  #   General Model                 #
58  # # # # # # # # # # # # # # # # # #
59  def DynamicModel(neurons=1, activation_func='sigmoid'):
60      """ A sequential Keras model that has an input layer, one
61          hidden layer with a dymanic number of units, and an output layer."""
62      model = Sequential()
63      model.add(Dense(neurons, input_dim=4, activation=activation_func, name='layer_1'))
64      model.add(Dense(3, activation='sigmoid', name='output_layer'))
65
66      # Don't change this!
67      model.compile(loss="categorical_crossentropy",
68                    optimizer="adam",
69                    metrics=['accuracy'])
70      return model
```

Build a Dynamic Model

→ Configure the model based on the parameters at hand.

NNs with Keras and Tensorflow

```
73  # (6) Evaluation + HyperParameter Search
74  # Below, we build KerasClassifiers using our model definitions. Use verbose=2 to see
75  # real-time updates for each epoch.
76  model = KerasClassifier(
77      build_fn=DynamiclineModel,
78      epochs=200,
79      batch_size=20,
80      verbose=0)
```

Note: Line 77 should say “DynamicModel”

Build a Single KerasClassifier

→ No changes in the model’s base parameters, i.e. epochs, batch_size.

NNs with Keras and Tensorflow

```
82  # (7) Define a set of unit numbers (i.e. "neurons") and activation functions
83  # that we want to explore.
84  param_grid = [
85      {
86          'activation_func': ['linear', 'sigmoid', 'relu', 'tanh'],
87          'neurons': [1, 5, 10, 15, 20, 25, 30]
88      }
89  ]
90
91  # (8)   Send the Keras model through GridSearchCV, and evaluate the performnce of every option in
92  #       param_grid for the "neuron" value.
93  grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
94  grid_result = grid.fit(X_train, y_train)
95
```

Define a param_grid and apply it with GridSearchCV.
→ Can include many hyperparameters, e.g. dropout.

NNs with Keras and Tensorflow

```
96    # (9) Print out a summarization of the results.
97    print("Best: %f using %s" % (grid_result.best_score_, grid_re
98    means = grid_result.cv_results_['mean_test_score']
99    stds = grid_result.cv_results_['std_test_score']
100   params = grid_result.cv_results_['params']
101   for mean, stdev, param in zip(means, stds, params):
102       print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.904762 using {'activation_func': 'linear', 'neurons': 25}
0.257143 (0.163299) with: {'activation_func': 'linear', 'neurons': 1}
0.628571 (0.129887) with: {'activation_func': 'linear', 'neurons': 5}
0.790476 (0.067343) with: {'activation_func': 'linear', 'neurons': 10}
0.780952 (0.211677) with: {'activation_func': 'linear', 'neurons': 15}
0.895238 (0.071270) with: {'activation_func': 'linear', 'neurons': 20}
0.904762 (0.058709) with: {'activation_func': 'linear', 'neurons': 25}
0.885714 (0.069985) with: {'activation_func': 'linear', 'neurons': 30}
0.276190 (0.094281) with: {'activation_func': 'sigmoid', 'neurons': 1}
0.523810 (0.304464) with: {'activation_func': 'sigmoid', 'neurons': 5}
0.600000 (0.141902) with: {'activation_func': 'sigmoid', 'neurons': 10}
0.704762 (0.071270) with: {'activation_func': 'sigmoid', 'neurons': 15}
0.695238 (0.058709) with: {'activation_func': 'sigmoid', 'neurons': 20}
0.695238 (0.105194) with: {'activation_func': 'sigmoid', 'neurons': 25}
0.790476 (0.074991) with: {'activation_func': 'sigmoid', 'neurons': 30}
0.342857 (0.185164) with: {'activation_func': 'relu', 'neurons': 1}
0.704762 (0.107750) with: {'activation_func': 'relu', 'neurons': 5}
0.780952 (0.110246) with: {'activation_func': 'relu', 'neurons': 10}
0.742857 (0.182201) with: {'activation_func': 'relu', 'neurons': 15}
0.895238 (0.035635) with: {'activation_func': 'relu', 'neurons': 20}
0.866667 (0.048562) with: {'activation_func': 'relu', 'neurons': 25}
0.876190 (0.058709) with: {'activation_func': 'relu', 'neurons': 30}
0.419048 (0.181203) with: {'activation_func': 'tanh', 'neurons': 1}
0.600000 (0.093314) with: {'activation_func': 'tanh', 'neurons': 5}
0.733333 (0.151785) with: {'activation_func': 'tanh', 'neurons': 10}
0.866667 (0.048562) with: {'activation_func': 'tanh', 'neurons': 15}
0.857143 (0.084112) with: {'activation_func': 'tanh', 'neurons': 20}
0.876190 (0.081927) with: {'activation_func': 'tanh', 'neurons': 25}
0.838095 (0.110246) with: {'activation_func': 'tanh', 'neurons': 30}
```

Print out the results.

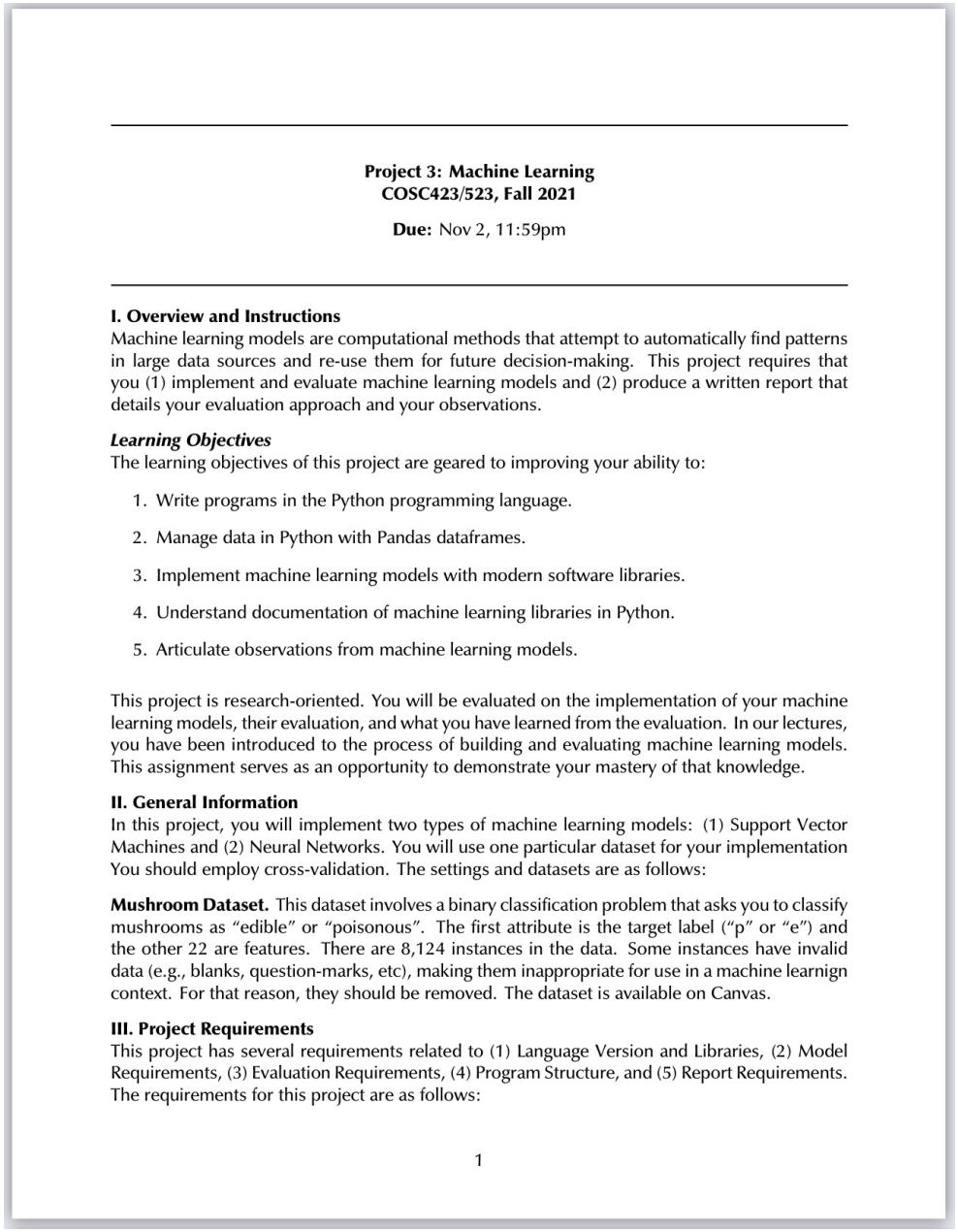
→ Linear activation + 25 neurons.

→ Personally surprised.

Exploration is a part of the process.

→ We don't know what "works" until after we explore.

Project 3: Machine Learning



Project 3: Machine Learning
COSC423/523, Fall 2021

Due: Nov 2, 11:59pm

I. Overview and Instructions
Machine learning models are computational methods that attempt to automatically find patterns in large data sources and re-use them for future decision-making. This project requires that you (1) implement and evaluate machine learning models and (2) produce a written report that details your evaluation approach and your observations.

Learning Objectives
The learning objectives of this project are geared to improving your ability to:

1. Write programs in the Python programming language.
2. Manage data in Python with Pandas dataframes.
3. Implement machine learning models with modern software libraries.
4. Understand documentation of machine learning libraries in Python.
5. Articulate observations from machine learning models.

This project is research-oriented. You will be evaluated on the implementation of your machine learning models, their evaluation, and what you have learned from the evaluation. In our lectures, you have been introduced to the process of building and evaluating machine learning models. This assignment serves as an opportunity to demonstrate your mastery of that knowledge.

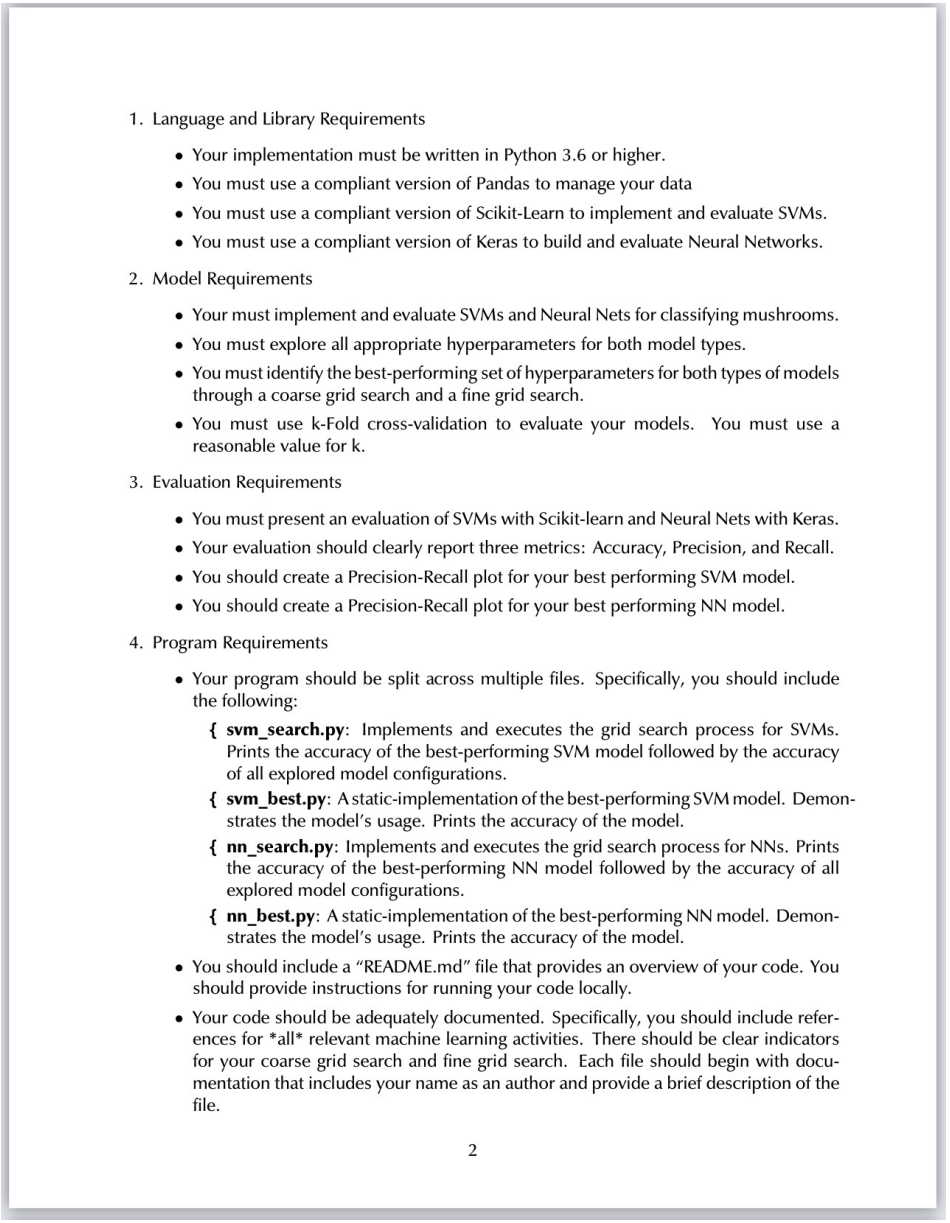
II. General Information
In this project, you will implement two types of machine learning models: (1) Support Vector Machines and (2) Neural Networks. You will use one particular dataset for your implementation You should employ cross-validation. The settings and datasets are as follows:

Mushroom Dataset. This dataset involves a binary classification problem that asks you to classify mushrooms as "edible" or "poisonous". The first attribute is the target label ("p" or "e") and the other 22 are features. There are 8,124 instances in the data. Some instances have invalid data (e.g., blanks, question-marks, etc), making them inappropriate for use in a machine learnign context. For that reason, they should be removed. The dataset is available on Canvas.

III. Project Requirements
This project has several requirements related to (1) Language Version and Libraries, (2) Model Requirements, (3) Evaluation Requirements, (4) Program Structure, and (5) Report Requirements. The requirements for this project are as follows:

1

Project 3: Machine Learning



1. Language and Library Requirements
   - Your implementation must be written in Python 3.6 or higher.
   - You must use a compliant version of Pandas to manage your data
   - You must use a compliant version of Scikit-Learn to implement and evaluate SVMs.
   - You must use a compliant version of Keras to build and evaluate Neural Networks.
2. Model Requirements
   - Your must implement and evaluate SVMs and Neural Nets for classifying mushrooms.
   - You must explore all appropriate hyperparameters for both model types.
   - You must identify the best-performing set of hyperparameters for both types of models through a coarse grid search and a fine grid search.
   - You must use k-Fold cross-validation to evaluate your models. You must use a reasonable value for k.
3. Evaluation Requirements
   - You must present an evaluation of SVMs with Scikit-learn and Neural Nets with Keras.
   - Your evaluation should clearly report three metrics: Accuracy, Precision, and Recall.
   - You should create a Precision-Recall plot for your best performing SVM model.
   - You should create a Precision-Recall plot for your best performing NN model.
4. Program Requirements
   - Your program should be split across multiple files. Specifically, you should include the following:
     - **svm_search.py**: Implements and executes the grid search process for SVMs. Prints the accuracy of the best-performing SVM model followed by the accuracy of all explored model configurations.
     - **svm_best.py**: A static-implementation of the best-performing SVM model. Demonstrates the model's usage. Prints the accuracy of the model.
     - **nn_search.py**: Implements and executes the grid search process for NNs. Prints the accuracy of the best-performing NN model followed by the accuracy of all explored model configurations.
     - **nn_best.py**: A static-implementation of the best-performing NN model. Demonstrates the model's usage. Prints the accuracy of the model.
   - You should include a "README.md" file that provides an overview of your code. You should provide instructions for running your code locally.
   - Your code should be adequately documented. Specifically, you should include references for *all* relevant machine learning activities. There should be clear indicators for your coarse grid search and fine grid search. Each file should begin with documentation that includes your name as an author and provide a brief description of the file.

2

Project 3: Machine Learning

5. Report Requirements

- You must include a two-page report on your project. The report must include your name, the name of the course, and the semester. For both types of models, your report should include the following sections:
  - **I. Introduction.** Describe an overview of the dataset and the problem you are solving.
  - **II. Pre-Processing.** Any steps taken to "clean" the dataset before being used for machine learning, e.g. removing rows with missing data / unusual characters.
  - **III. Grid Search.** Articulate the explored hyperparameter configurations that your program evaluated and what was observed. Specifically report how k-Fold cross-validation was employed.
  - **IV. Best-Performing Models.** Describe the configuration of your best performing model and their accuracy. You should include both Precision-Recall plots for your best-performing models.
- You should conclude your written report with a section named "**V. Conclusion**" in which you briefly summarize what you learned from your process. Specifically state which type of model that you believe to be more effective.
- The report must be submitted as a PDF file.

IV. Grading and Submission

The assignment is worth 100 points. Create a ZIP file that includes all of your files. Upload the ZIP file to the Project 3 submission folder on Canvas by the due date. For questions regarding the late policy for assignment submission, please consult the syllabus.

The following grading scheme will be used for all students:

Requirement	Point Value
README exists and follows specification	10 points
Code is documented following specification	10 points
Model Requirements	10 points
Evaluation Requirements	20 points
Program Requirements	25 points
Report Requirements	25 points
Total	100

Note I: Failure to meet the Language and Library Requirements will result in an automatic grade of zero.

Note II: You should submit only the required files: 4 Python files, 1 Readme file, and your PDF report.

3

Next Time

Reasoning Under Uncertainty