MSSP 608 Recitation 2

Slides by Chetan Parthiban

What do all these imports do

```
[ ] import math
    import pandas as pd
    import numpy as np
    import statsmodels.formula.api as smf
    import matplotlib.pyplot as plt
    import calendar
    import itertools
    from scipy import stats
    from matplotlib import dates
    from datetime import datetime

    from sklearn.metrics import accuracy_score, precision_score, recall_score, cohen_kappa_score, confusion_matrix, ConfusionMatrixDisplay
    from sklearn.linear_model import LogisticRegression
    from sklearn.tree import DecisionTreeClassifier
    from sklearn import tree
    from sklearn.naive_bayes import MultinomialNB
    from sklearn.model_selection import train_test_split, KFold, GridSearchCV
    from sklearn.utils.testing import ignore_warnings
    from sklearn.exceptions import ConvergenceWarning, UndefinedMetricWarning
```

3

Use third party libraries if possible

Almost anytime you want to do something, you probably want to use someone else's code to do it. In this case, whenever you're working with graphs in Python, you probably want to use NetworkX.

Then your code is as simple as this (requires scipy):

```
import networkx as nx
```

prolly use someone else's code

Let's break this down into parts

```
[ ]
# Python Standard Library Imports
import math
import calendar
import itertools
from datetime import datetime

# Basic Data Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import dates

# Compute Useful Statistics
from scipy import stats
import statsmodels.formula.api as smf

# Most of our Machine Learning Stuff
from sklearn.metrics import accuracy_score, precision_score, recall_score, cohen_kappa_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split, KFold, GridSearchCV
from sklearn.utils.testing import ignore_warnings
from sklearn.exceptions import ConvergenceWarning, UndefinedMetricWarning
```

Python Standard Library

```python
# Python Standard Library Imports
import math
import calendar
import itertools
from datetime import datetime
```

- Set of useful functions/classes that are included in python straight out of the box
- **Math:** contains useful math operations and constants (think sin or cos or pi)
- **Calendar/Datetime:** provide a convenient interface to work with dates (for example, if we want to extract the month/day/year from a string)
- **Itertools**: provides tools to work with python iterables (for example, if we want to get all combinations of elements from two lists we can use itertools.product)

Basic Data Libraries

```python
# Basic Data Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import dates
```

- These libraries have a lot of the commonly used data-science functionalities already worked out (and are much faster than implementing them yourselves.
- **Pandas:** awesome library to work with tabular data. Allows us to store different types of data together in an organized way and easily save/load data with files.
- **Numpy:** probably the most popular linear algebra library in python. This library is very useful whenever we are working with matrices of numbers. It has many useful functions like the inverse, eigenvalue decompositions, or even calculating the mean/variance.
- **Matplotlib**: Quick library for making plots in python

More Data Libraries

```python
# Compute Useful Statistics
from scipy import stats
import statsmodels.formula.api as smf
```

- These libraries are less commonly used then the previous set, but extend their functionality
- **Scipy:** an extension of numpy to have even more functionality. "if it's covered in a general textbook on numerical computing, it's probably implemented in SciPy"
- **Statsmodels**: allows us to estimate statistical models and perform commons statistical tests

Scikit-Learn

```python
# Most of our Machine Learning Stuff
from sklearn.metrics import accuracy_score, precision_score, recall_score, cohen_kappa_score, confusion_matrix, ConfusionMa
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split, KFold, GridSearchCV
from sklearn.utils.testing import ignore_warnings
from sklearn.exceptions import ConvergenceWarning, UndefinedMetricWarning
```

- Scikit-Learn contains most of the machine learning functionality that we could want
- Notice that this is most of our imports
- The great part about scikit-learn is that we can do a lot of experiments doing basically the same (small amount of) code

```python
# Decide on our number of folds for cross validation in the hyerparameter search
kfolds = 5

# Initialize a search using cross validation in sklearn
search = GridSearchCV(estimator=DecisionTreeClassifier(random_state=123),
                      param_grid=hyperparameters, cv=kfolds, scoring="accuracy")

# Train a classifier with each combination of hyperparameters and take the best one
# and print out the results
classifier = search.fit(X_train, y_train)
accuracy = classifier.best_score_
best_fit = classifier.best_estimator_
print(f"Best fit when training was {best_fit}\nWith {100*accuracy:.1f}% accuracy.")

# Evaluate our best model's performance on the test set and print the results
y_pred = classifier.predict(X_test)
accuracy = 100*accuracy_score(y_test, y_pred)
print(f"Accuracy on held-out test set: {accuracy:.1f}%")
```

Hyperparameter Optimization

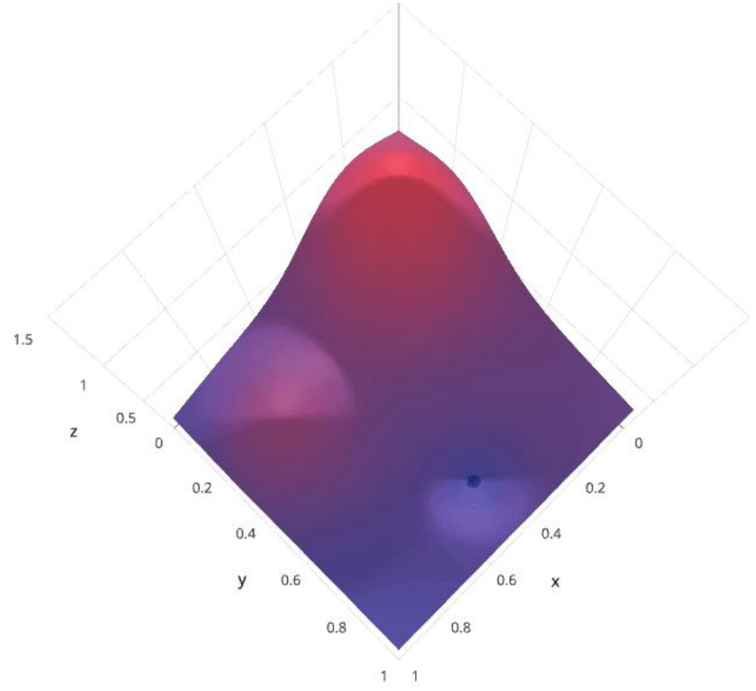
Brief Review of Hyperparameter Optimization

Core Ideas

1. When defining the model we make a LOT of decisions (examples: type of regularization, amount of regularization, etc.)
   a) We call these non-learned parameters **hyperparameters**
   b) These hyperparameters can have a large impact on our final learned model
2. While we can make some educated guesses, we can also optimize these parameters for our dataset by just trying a bunch of them out and picking the best one

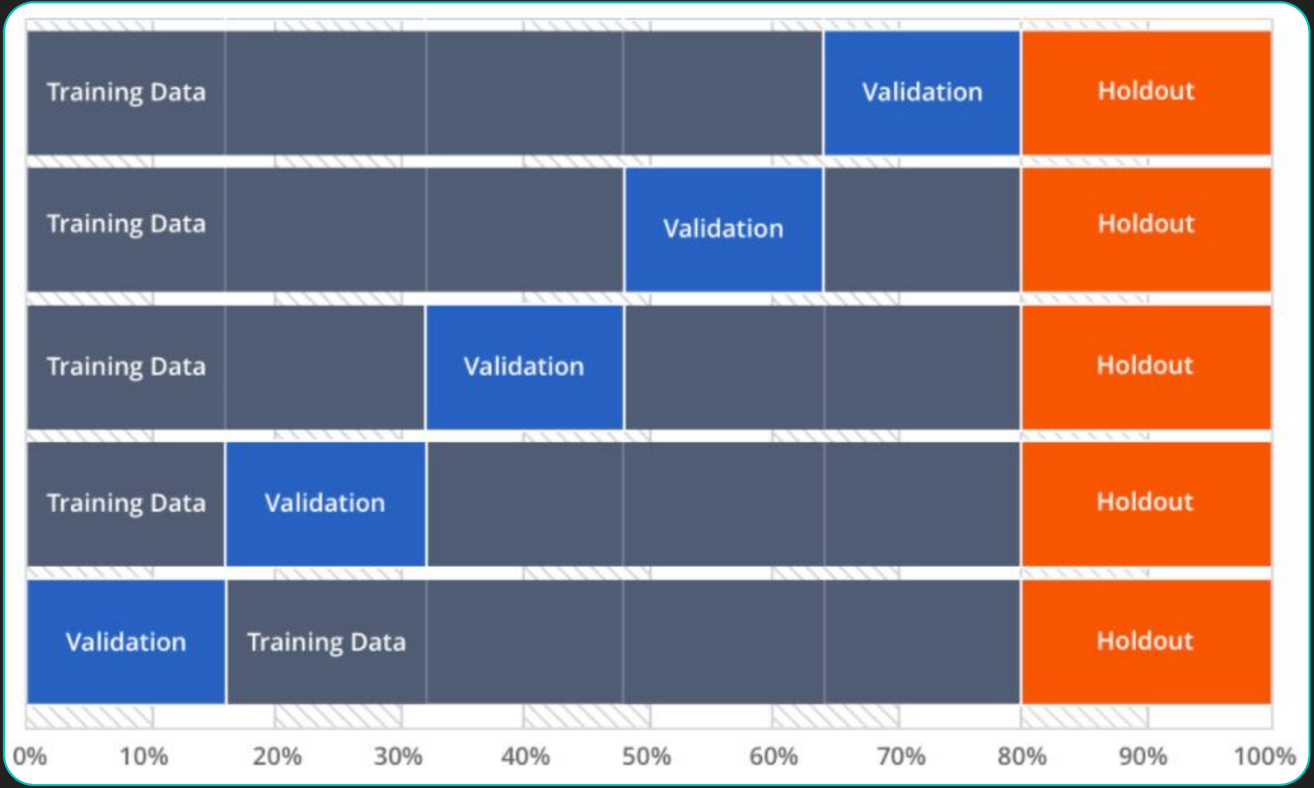
Ways to go about this

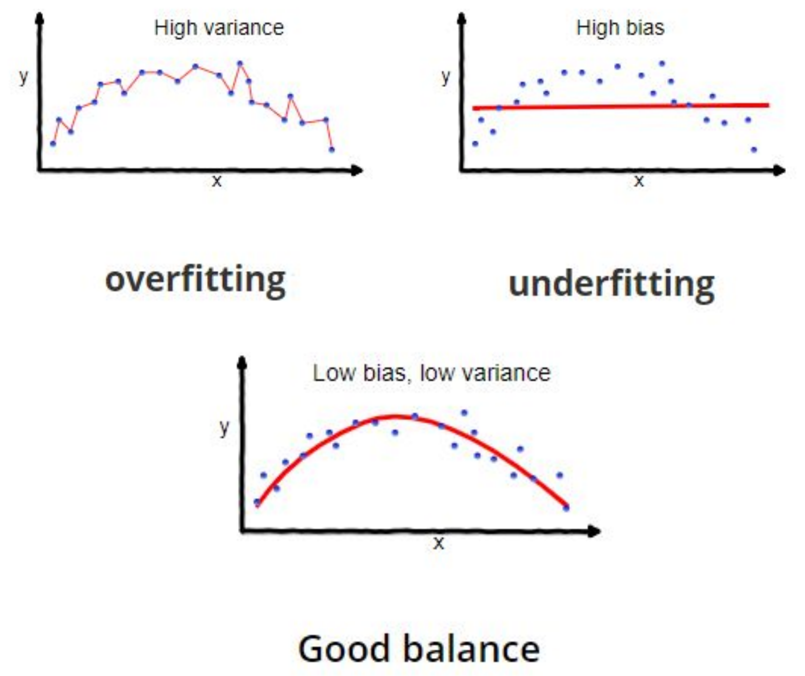
1. **Grid Search:** create a grid of all possible hyperparameter combinations, then test each point in the grid and test it – The most exhaustive but also the most expensive
2. **Random Grid Search:** randomly sample points from the grid of hyperparameters and test these random points – Cheaper but you may not get all the way to the true best
3. **Other Clever Methods**: for example, exponential search, where we start with a big search area and keep trying to cut it in half

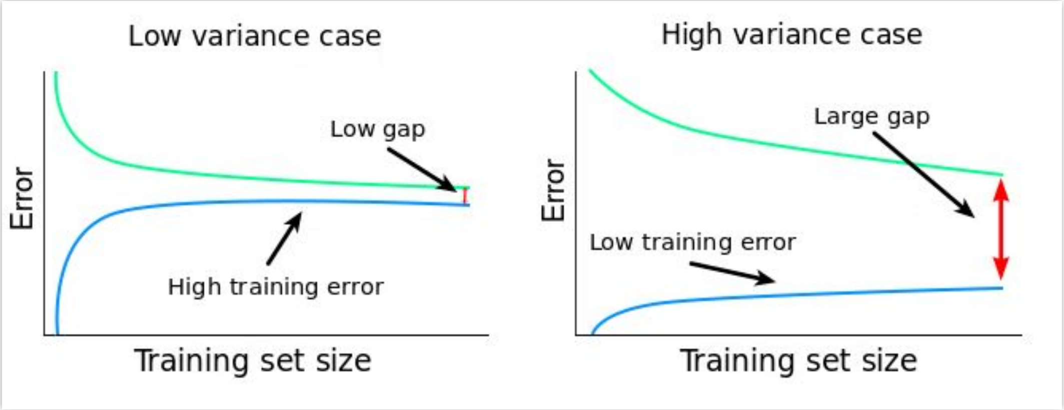


Dangers of Hyperparameter Optimization

- We usually evaluate our model's performance based on the test set
- If we do not have a special test set (usually called the **validation** set) just for hyperparameter optimization, we can overfit to the test set
- When we optimize over test set and not a validation set, we are leaking the test set to our model
- Also, it can just take a super long time. Let's say I have 6 hyperparameters each with 10 values I want to test. Then grid search would require me to train 10^6 = 1,000,000 models!







Bias/Variance Tradeoff

More Bias/Variance Tradeoff