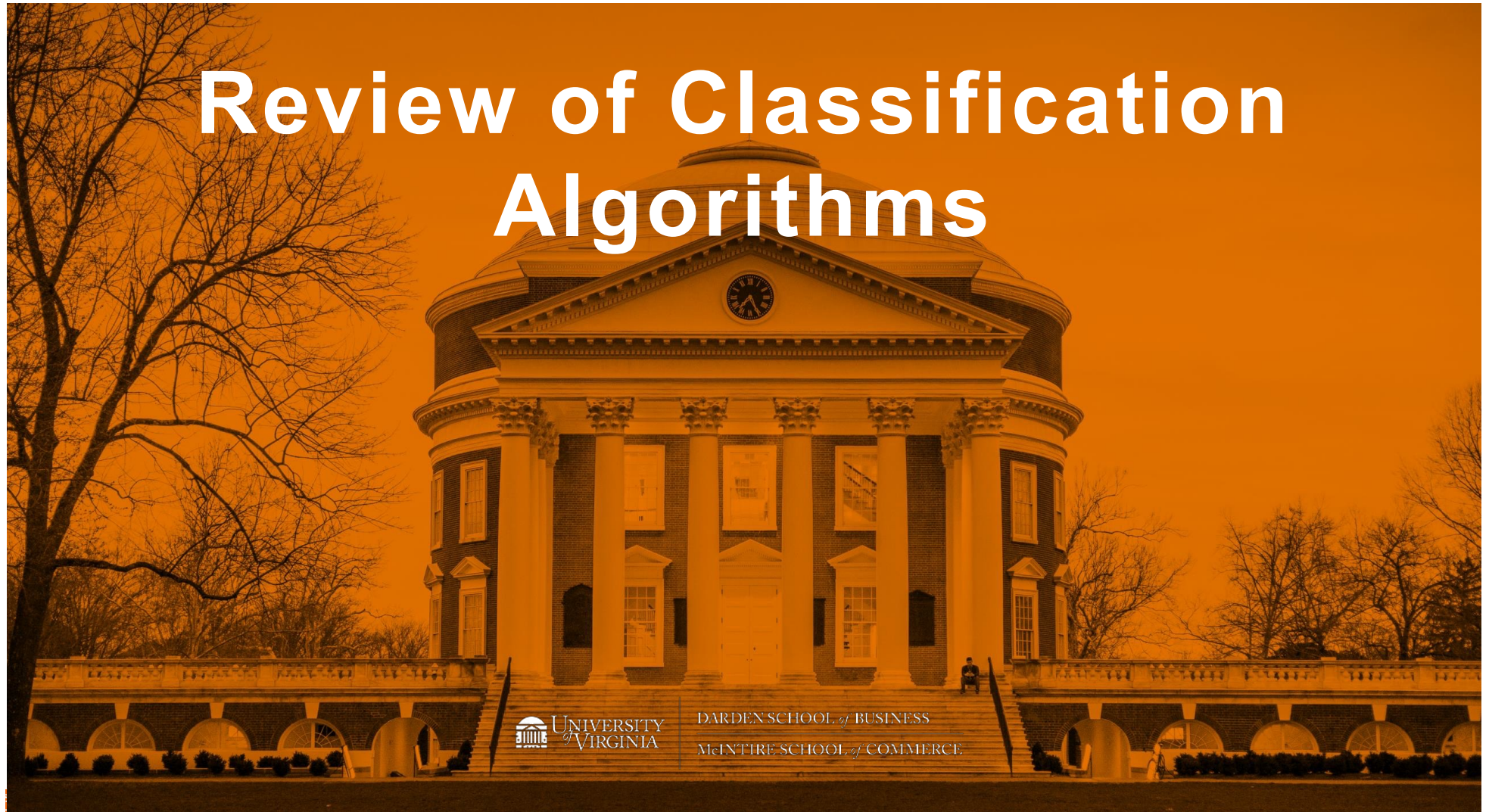


Review of Classification Algorithms



Log Loss

The negative average of corrected probabilities is the Log loss or Binary cross-entropy:

$$- \frac{1}{N} \sum_{i=1}^N (\log(p_i))$$

ID	Actual	Predicted probabilities	Corrected Probabilities	Log
ID6	1	0.94	0.94	-0.0268721464
ID1	1	0.90	0.90	-0.0457574906
ID7	1	0.78	0.78	-0.1079053973
ID8	0	0.56	0.44	-0.3565473235
ID2	0	0.51	0.49	-0.30980392
ID3	1	0.47	0.47	-0.3279021421
ID4	1	0.32	0.32	-0.4948500217
ID5	0	0.10	0.90	-0.0457574906



Take the
average
and
multiply
by -




Log Loss





SKLearn Models

[Install](#) [User Guide](#) [API](#) [Examples](#) [Community](#) [More](#) [Go](#)

scikit-learn

Machine Learning in Python

[Getting Started](#) [Release Highlights for 1.1](#) [GitHub](#)

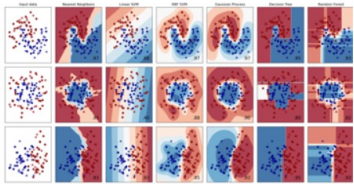
- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.

Algorithms: SVM, nearest neighbors, random forest, and more...

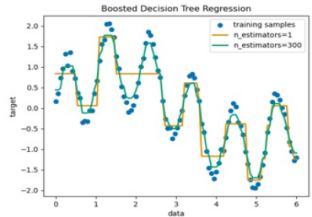


Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, nearest neighbors, random forest, and more...

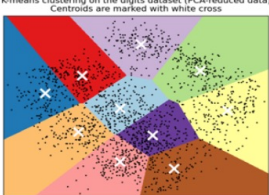


Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, and more...



SKLearn Models

The complete list of models are available here:

https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

To be able to use them, we should import them, and then use the correct method while training the model.



Trying Different Algorithms

1. Notebook Styling and Package Management

```
: import numpy as np # Library for math operations
import pandas as pd # Library for data handling
import sklearn # The machine learning library we will be using in this entire course
from sklearn import tree # Tree function is used for visualizing decision tree
from sklearn.metrics import * # Importing function that can be used to calculate different metrics
from sklearn.tree import DecisionTreeClassifier # Importing Decision Tree Classifier
from sklearn.model_selection import train_test_split # Importing function that can split a dataset into training
from sklearn.preprocessing import MinMaxScaler # Importing function for scaling the data
from sklearn.preprocessing import LabelEncoder # Importing function for processing the labels
from sklearn.ensemble import GradientBoostingClassifier # Importing GB Classifier
from sklearn.model_selection import GridSearchCV # Importing GridSearchCV
from sklearn.model_selection import RandomizedSearchCV # Importing RandomSearchCV
from xgboost import XGBClassifier # Importing the XGBoost Classifier
import matplotlib.pyplot as plt # Importing the package for plotting
plt.style.use('fivethirtyeight') # Use the styling from FiveThirtyEight Website
import seaborn as sns # Importing another package for plotting
```

```
clf = DecisionTreeClassifier(random_state = 1)
clf.fit(trainData, trainLabels)
```



Trying Different Algorithms- Example: Random Forest

```
import numpy as np # Library for math operations
import pandas as pd # Library for data handling
import sklearn # The machine learning library we will be using in this entire course
from sklearn import tree # Tree function is used for visualizing decision tree
from sklearn.metrics import * # Importing function that can be used to calculate different metrics
from sklearn.tree import DecisionTreeClassifier # Importing Decision Tree Classifier
from sklearn.ensemble import RandomForestClassifier # Importing Random Forest Classifier
from sklearn.model_selection import train_test_split # Importing function that can split a dataset into training
from sklearn.preprocessing import MinMaxScaler # Importing function for scaling the data
from sklearn.preprocessing import LabelEncoder # Importing function for processing the labels
from sklearn.ensemble import GradientBoostingClassifier # Importing GB Classifier
from sklearn.model_selection import GridSearchCV # Importing GridSearchCV
from sklearn.model_selection import RandomizedSearchCV # Importing RandomSearchCV
from xgboost import XGBClassifier # Importing the XGBoost Classifier
import matplotlib.pyplot as plt # Importing the package for plotting
plt.style.use('fivethirtyeight') # Use the styling from FiveThirtyEight Website
import seaborn as sns # Importing another package for plotting
```

```
clf = RandomForestClassifier(random_state = 1)
clf.fit(trainData, trainLabels)
```



HyperParameter Tuning for Different Algorithms

Each algorithm has its own set of parameters that need to be tuned. Typically, we tune a few important parameters, called hyperparameters.

Before tuning for hyperparameters, we should do a little research to identify them. The documentations of the algorithms also provide some insights:

1.11.2.3. Parameters

The main parameters to adjust when using these methods is `n_estimators` and `max_features`. The former is the number of trees in the forest. The larger the better, but also the longer it will take to compute. In addition, note that results will stop getting significantly better beyond a critical number of trees. The latter is the size of the random subsets of features to consider when splitting a node. The lower the greater the reduction of variance, but also the greater the increase in bias. Empirical good default values are `max_features=1.0` or equivalently `max_features=None` (always considering all features instead of a random subset) for regression problems, and `max_features="sqrt"` (using a random subset of size `sqrt(n_features)`) for classification tasks (where `n_features` is the number of features in the data). The default value of `max_features=1.0` is equivalent to bagged trees and more randomness can be achieved by setting smaller values (e.g. 0.3 is a typical default in the literature). Good results are often achieved when setting `max_depth=None` in combination with `min_samples_split=2` (i.e., when fully developing the trees). Bear in mind though that these values are usually not optimal, and might result in models that consume a lot of RAM. The best parameter values should always be cross-validated. In addition, note that in random forests, bootstrap samples are used by default (`bootstrap=True`) while the default strategy for extra-trees is to use the whole dataset (`bootstrap=False`). When using bootstrap sampling the generalization error can be estimated on the left out or out-of-bag samples. This can be enabled by setting `oob_score=True`.

Note: The size of the model with the default parameters is $O(M * N * \log(N))$, where M is the number of trees and N is the number of samples. In order to reduce the size of the model, you can change these parameters: `min_samples_split`, `max_leaf_nodes`, `max_depth` and `min_samples_leaf`.



Trying Different Algorithms- Exercise: Support Vector Machines

Now, go to McAfee_PredictiveModel.ipynb and use SVM to train the model. What hyperparameters you tuned for? What is the AUC for the model?



