Predictive Modeling Project Template

1. How to structure an end-to-end predictive modeling project.

2. How to map the tasks you learned about in Part II onto a project.

3. How to best use the structured project template to ensure an accurate result for your dataset.

- Working through machine learning problems from end-to-end is critically important.
- You can read about machine learning.
- You can also try out small one-o recipes.
- But applied machine learning will not come alive for you until you work through a dataset from beginning to end.

Use A Structured Step-By-Step Process

1. Define Problem.
2. Summarize Data.
3. Prepare Data.
4. Define Problem.
5. Summarize Data.
6. Prepare Data.

Machine Learning Project Template in Python

- This lecture presents a project template that you can use to work through machine learning problems in Python end-to-end.
- Please refer to the Template Summary in Chapter 18

How To Use The Project Template

1. Create a new file for your project (e.g. project name.py).

2. Copy the project template.

3. Paste it into your empty project file.

4. Start to fill it in, using recipes from this class and other resources

Prepare Problem

This step is about loading everything you need to start working on your problem. This includes:

ˆ Python modules, classes and functions that you intend to use.

ˆ Loading your dataset from CSV.

Summarize Data

This step is about better understanding the data that you have available. This includes understanding your data using:

ˆ Descriptive statistics such as summaries.

ˆ Data visualizations such as plots with Matplotlib, ideally using convenience functions from Pandas.

Prepare Data

This step is about preparing the data in such a way that it best exposes the structure of the problem and the relationships between your input attributes with the output variable. This includes tasks such as:

ˆ Cleaning data by removing duplicates, marking missing values and even imputing missing values.

ˆ Feature selection where redundant features may be removed and new features developed.

ˆ Data transforms where attributes are scaled or redistributed in order to best expose the structure of the problem later to learning algorithms.

Evaluate Algorithms

This step is about finding a subset of machine learning algorithms that are good at exploiting the structure of your data (e.g. have better than average skill). This involves steps such as:

ˆ Separating out a validation dataset to use for later con rmation of the skill of your developed model.

ˆ Defining test options using scikit-learn such as cross-validation and the evaluation metric to use.

ˆ Spot-checking a suite of linear and nonlinear machine learning algorithms.

ˆ Comparing the estimated accuracy of algorithms.

Improve Accuracy

Once you have a shortlist of machine learning algorithms, you need to get the most out of them.

There are two different ways to improve the accuracy of your models:

ˆ Search for a combination of parameters for each algorithm using scikit-learn that yields the best results.

ˆ Combine the prediction of multiple models into an ensemble prediction using ensemble techniques.

Finalize Model

Once you have found a model that you believe can make accurate predictions on unseen data, you are ready to finalize it. Finalizing a model may involve sub-tasks such as:

ˆ Using an optimal model tuned by scikit-learn to make predictions on unseen data.

ˆ Creating a standalone model using the parameters tuned by scikit-learn.

ˆ Saving an optimal model to a file for later use.

Tips For Using The Template Well

This section lists tips that you can use to make the most of the machine learning project template in Python.

ˆ Fast First Pass. Make a first-pass through the project steps as fast as possible. This will give you confidence that you have all the parts that you need and a baseline from which to improve.

ˆ Cycles. The process in not linear but cyclic. You will loop between steps, and probably spend most of your time in tight loops between steps 3-4 or 3-4-5 until you achieve a level of accuracy that is su cient or you run out of time.

ˆ Attempt Every Step. It is easy to skip steps, especially if you are not con dent or familiar with the tasks of that step. Try and do something at each step in the process, even if it does not improve accuracy. You can always build upon it later. Don't skip steps, just reduce their contribution.

ˆ Ratchet Accuracy. The goal of the project is model accuracy. Every step contributes towards this goal. Treat changes that you make as experiments that increase accuracy as the golden path in the process and reorganize other steps around them. Accuracy is a ratchet that can only move in one direction (better, not worse).

ˆ Adapt As Needed. Modify the steps as you need on a project, especially as you become more experienced with the template. Blur the edges of tasks, such as steps 4-5 to best serve model accuracy.

Summary

- In this lesson you discovered a machine learning project template in Python.
- I laid out the steps of a predictive modeling machine learning project with the goal of maximizing model accuracy.
- You can copy-and-paste the template and use it to jump-start your current or next machine learning project in Python.

Final Project

- You need to see how all of the pieces of a predictive modeling machine learning project actually put together.
- In this final project/lesson you will complete your first machine learning project using Python.
-

In this step-by-step lab project you will:

ˆ Download and install Python SciPy and get the most useful package for machine learning in Python.

ˆ Load a dataset and understand its structure using statistical summaries and data visual-ization.

ˆ Create 6 machine learning models, pick the best and build confidence that the accuracy is reliable.

Flowers

- ˆ Attributes are numeric so you do not have to figure out how to load and handle data.
- ˆ It is a Classification problem, allowing you to practice with an easier type of supervised learning algorithm.
- ˆ It is a multiclass Classification problem (multi-nominal) that may require some specialized handling.
- ˆ It only has 4 attributes and 150 rows, meaning it is small and easily fits into memory (and a screen or single sheet of paper).
- ˆ All of the numeric attributes are in the same units and the same scale not requiring any special scaling or transforms to get started.

What we are going to do

1. Loading the dataset.
2. Summarizing the dataset.
3. Visualizing the dataset.
4. Evaluating some algorithms.
5. Making some predictions.

Import libraries

```
# Load libraries
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

Load Dataset

```
# Load dataset
filename = 'iris.data.csv'
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(filename, names=names)
```

Summarize the Dataset

Now it is time to take a look at the data. In this step we are going to take a look at the data a few different ways:

- Dimensions of the dataset.
- Peek at the data itself.
- Statistical summary of all attributes.
- Breakdown of the data by the class variable.

```
# shape
print(dataset.shape)
# head
print(dataset.head(20))
 # descriptions
 print(dataset.describe())
# class distribution
print(dataset.groupby('class').size())
```

Data Visualizations

We now have a basic idea about the data. We need to extend this with some visualizations. We are going to look at two types of plots:

ˆ Univariate plots to better understand each attribute.

ˆ Multivariate plots to better understand the relationships between attributes.

Univariate Plots

```
# box and whisker plots
dataset.plot(kind=box, subplots=True, layout=(2,2), sharex=False, sharey=False)
pyplot.show()
#  histograms
dataset.hist()
pyplot.show()
#  scatter plot matrix
scatter_matrix(dataset) pyplot.show()
```

Evaluate Some Algorithms

Now it is time to create some models of the data and estimate their accuracy on unseen data.

Here is what what I want you to cover in this step:

1. Separate out a validation dataset.
2. Setup the test harness to use 10-fold cross-validation.
3. Build 5 different models to predict species from flower measurements
4. Select the best model.

Create a Validation Dataset

```
#  Split-out validation dataset
array = dataset.values
X = array[:,0:4] Y = array[:,4] validation_size = 0.20 seed =
7

X_train, X_validation, Y_train, Y_validation = train_test_split(X, Y, test_size=validation_size,
      random_state=seed)
```

Test Harness

- We will use 10-fold cross-validation to estimate accuracy on unseen data.
- This will split our dataset into 10 parts, e.g. the model will train on 9 and test on 1 and repeat for all combinations of train-test splits.
- We are using the metric of accuracy to evaluate models. This is a proportion of the number of correctly predicted instances divided by the total number of instances in the dataset multiplied by 100 to give a percentage (e.g. 95% accurate).
- We will be using the scoring variable when we run build and evaluate each model next.

Build Models

- Logistic Regression (LR).
- Linear Discriminant Analysis (LDA).
- k-Nearest Neighbors (KNN).
- Classification and Regression Trees (CART).
- Gaussian Naive Bayes (NB).
- Support Vector Machines (SVM).

```
#   Spot-Check Algorithms
models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis())) models.append(('KNN',
KNeighborsClassifier())) models.append(('CART', DecisionTreeClassifier())) models.append(('NB',
GaussianNB())) models.append(('SVM', SVC(gamma='auto')))
#   evaluate each model in turn
results = []
names = []
for name, model in models:
   kfold = KFold(n_splits=10, random_state=seed)
   cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accuracy')
   results.append(cv_results)
   names.append(name)
   msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
   print(msg)
```

Select The Best Model

```
LR: 0.966667 (0.040825)
LDA: 0.975000 (0.038188)
KNN: 0.983333 (0.033333)
CART: 0.975000 (0.038188)
NB: 0.975000 (0.053359)
SVM: 0.981667 (0.025000)
```

```
#  Compare Algorithms
fig = pyplot.figure() fig.suptitle('Algorithm Comparison') ax =
fig.add_subplot(111) pyplot.boxplot(results) ax.set_xticklabels(names)
pyplot.show()
```

Make Predictions

- The KNN algorithm was the most accurate model that we tested.
- Now we want to get an idea of the accuracy of the model on our validation dataset.
- This will give us an independent nal check on the accuracy of the best model.
- It is important to keep a validation set just in case you made a slip during training, such as over tting to the training set or a data leak.
- Both will result in an overly optimistic result.
- We can run the KNN model directly on the validation set and summarize the results as a final accuracy score, a confusion matrix and a Classification report.

```
#  Make predictions on validation dataset
knn = KNeighborsClassifier() knn.fit(X_train, Y_train)
predictions = knn.predict(X_validation)

print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

Summary

- In this lesson you discovered step-by-step how to complete your first machine learning project in Python.
- You discovered that completing a small end-to-end project from loading the data to making predictions is the best way to get familiar with the platform.