Section 01

1.1 Training a Decision Tree Classifer

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import datasets


iris = datasets.load_iris()

fearures = iris.data
target = iris.target


decisiontree = DecisionTreeClassifier(random_state=0)


model = decisiontree.fit(fearures, target)


observation = [[5,4,3,2]]


model.predict(observation)
```

```
array([1])
```

```
model.predict_proba(observation)
```

```
array([[0., 1., 0.]])
```

1.2 Controlling the tree size

```
decisiontree = DecisionTreeClassifier(random_state = 0,
                                      max_depth = None,
                                      min_samples_split = 2,
                                      min_samples_leaf = 2,
                                      max_leaf_nodes = 3,
                                      min_impurity_decrease = 0)


model = decisiontree.fit(fearures, target)
```

Section 02

2.1 Creating a K-Nearest Neighbor Classifier

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn import datasets


iris = datasets.load_iris()
x = iris.data
y = iris.target


standardizer = StandardScaler()


x_std = standardizer.fit_transform(x)


knn = KNeighborsClassifier(n_neighbors=5, n_jobs=-1).fit(x_std, y)


new_observations = [[0.75,0.75,0.75,0.75],
                    [1,1,1,1]]


knn.predict(new_observations)
```

```
array([1, 2])
```

Section 03

3.1 Training a Binary Classifier

```
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.preprocessing import StandardScaler


iris = datasets.load_iris()
features = iris.data[:100,:]
target = iris.target[:100]
```

```
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)
```

```
logistic_regression = LogisticRegression(random_state=0)
```

```
model = logistic_regression.fit(features_standardized, target)
```

```
new_observations = [[0.5,0.5,0.5,0.5]]
```

```
model.predict(new_observations)
```

```
array([1])
```

```
model.predict_proba(new_observations)
```

```
array([[0.17738424, 0.82261576]])
```

3.2 Training a Multiclass Classifier

```
iris = datasets.load_iris()
features = iris.data
target = iris.target
```

```
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)
```

```
logistic_regression = LogisticRegression(random_state=0, multi_class="ovr")
```

```
model = logistic_regression.fit(features_standardized, target)
```

3.3 Reducing Variance Through Regularization

```
from sklearn.linear_model import LogisticRegressionCV
```

```
logistic_regression = LogisticRegressionCV(penalty='l2', Cs=10, random_state=0, n
```

```
model = logistic_regression.fit(features_standardized, target)
```

3.4 Handling Imbalanced Classes

```
import numpy as np
from sklearn.linear_model import LogisticRegression


iris = datasets.load_iris()
features = iris.data
target = iris.target


features = features[40:,:]
target = target[40:]


target = np.where((target==0), 0, 1)


scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)


logistic_regression = LogisticRegression(random_state=0, class_weight="balanced")


model = logistic_regression.fit(features_standardized, target)
```

Section 04

4.1 Training a Linear Classifier

```
from sklearn.svm import LinearSVC


iris = datasets.load_iris()
features = iris.data[:100,:2]
target = iris.target[:100]


features_standardized = scaler.fit_transform(features)


svc = LinearSVC(C=1.0)


model = svc.fit(features_standardized, target)
```

```
new_observations = [[-2, 3]]
```

```
svc.predict(new_observations)
```

```
array([0])
```

4.2 Creating Predicted Probabilites

```
from sklearn.svm import SVC
```

```
iris = datasets.load_iris()
features = iris.data
target = iris.target
```

```
features_standardized = scaler.fit_transform(features)
```

```
svc = SVC(kernel="linear", probability=True, random_state=0)
```

```
model = svc.fit(features_standardized, target)
```

```
new_observations = [[.4,.4,.4,.4]]
```

```
model.predict_proba(new_observations)
```

```
array([[0.00623541, 0.96973799, 0.0240266 ]])
```

4.3 Intentifying Support Vectors

```
iris = datasets.load_iris()
features = iris.data[:100,:]
target = iris.target[:100]
```

```
features_standardized = scaler.fit_transform(features)
```

```
svc = SVC(kernel="linear", random_state=0)
```

```
model = svc.fit(features_standardized, target)
```

```
model.support_vectors_
```

```
array([[-0.5810659 ,  0.42196824, -0.80497402, -0.50860702],
       [-1.52079513, -1.67737625, -1.08231219, -0.86427627],
       [-0.89430898, -1.4674418 ,  0.30437864,  0.38056609],
       [-0.5810659 , -1.25750735,  0.09637501,  0.55840072]])
```

Section 05

5.1 Training a Classifier for contious features

```
from sklearn.naive_bayes import GaussianNB


iris = datasets.load_iris()
features = iris.data
target = iris.target


classifier = GaussianNB()


model = classifier.fit(features, target)


new_observations = [[4,4,4,.4]]


model.predict(new_observations)
```

```
array([1])
```

5.2 Training a classifier for discrete and count features

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer


text_data = np.array(['I love Brazil. Brazil!',
                      'Brazil is best',
                      'Germany beats both'])


count = CountVectorizer()
bag_of_words = count.fit_transform(text_data)


features = bag_of_words.toarray()
```

```
features = bag_of_words.toarray()
```

```
features
```

```
array([[0, 0, 0, 2, 0, 0, 1],
       [0, 1, 0, 1, 0, 1, 0],
       [1, 0, 1, 0, 1, 0, 0]])
```

```
target = np.array([0,0,1])
```

```
classifier = MultinomialNB(class_prior=[0.25, 0.5])
```

```
model = classifier.fit(features, target)
```

```
new_observations = [[0,0,0,1,0,1,0]]
```

```
model.predict(new_observations)
```

```
array([0])
```

5.3 Training a Naive Bayes classifier for Binary Features

```
from sklearn.naive_bayes import BernoulliNB
```

```
features = np.random.randint(2, size=(100, 3))
```

```
target = np.random.randint(2, size=(100,1)).ravel()
```

+ Code + Text

```
classifier = BernoulliNB(class_prior=[0.25, 0.5])
```

```
model = classifier.fit(features, target)
```

✓ 0s completed at 14:16