A sneak peak into Artificial Intelligence

and Machine Learning

Cosa vedrete:

1. Cos'è l'intelligenza artificiale
2. Perchè usare l'intelligenza artificiale
3. Modelli di machine learning: in particolare Decision Trees (e la versione di ensemble, RandomForest) e Neural Networks
4. Esempi

Note: le immagini e alcuni degli esempi mostrati sono stati presi dal libro "Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow. Concepts, Tools, and Techniques to Build Intelligent Systems" di Aurélien Géron

Cos'è l'intelligenza artificiale

"Machine Learning is the field of study that gives computers the ability to **learn** without being explicitly programmed"

"A computer program is said to learn from **experience** E with respect to some **task** T and some **performance measure** P, if its performance on T, as measured by P, improves with experience E."

Come lavoriamo...

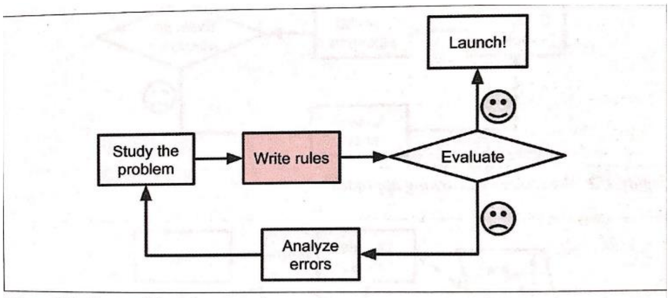


Figure 1-1. The traditional approach

...come possiamo lavorare...

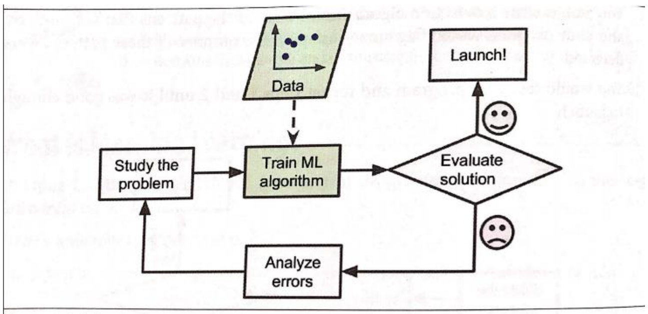


Figure 1-2. The Machine Learning approach

...o ancora meglio...

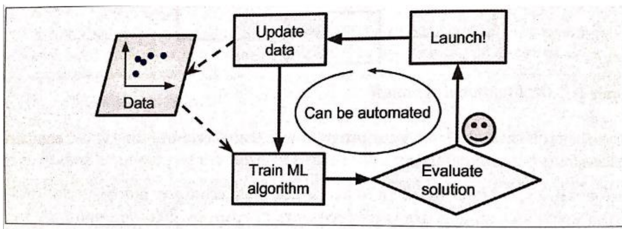


Figure 1-3. Automatically adapting to change

Perchè usare l'intelligenza artificiale

- Problems for which existing solutions require a **lot of fine-tuning** or **long list of rules**
- **Complex problems** for which using a traditional approach yields **no good solution**
- **Fluctuating environments**
- Getting insights about complex problems and **large amounts of data**

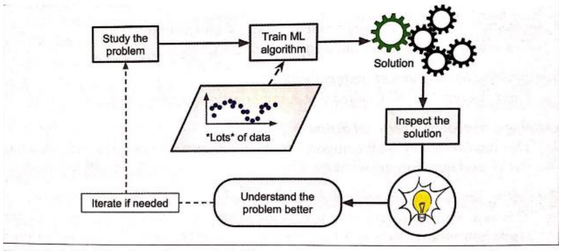


Figure 1-4. Machine Learning can help humans learn

In concreto:

Image Classification problems:

- Analyzing images of products on a production line to automatically classify them
- Detecting tumors in brain scans

Using CNNs

Text Classification and Sentiment Analysis problems:

- Automatically classifying news articles
- Automatically flagging offensive comments on discussion forums
- Summarizing long documents automatically
- Creating a chatbot or a personal assistant

Using NLP tools i.e. RNNs or CNNs

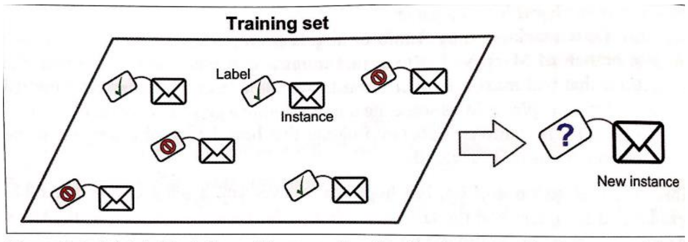


Figure 1-5. A labeled training set for spam classification (an example of supervised learning)

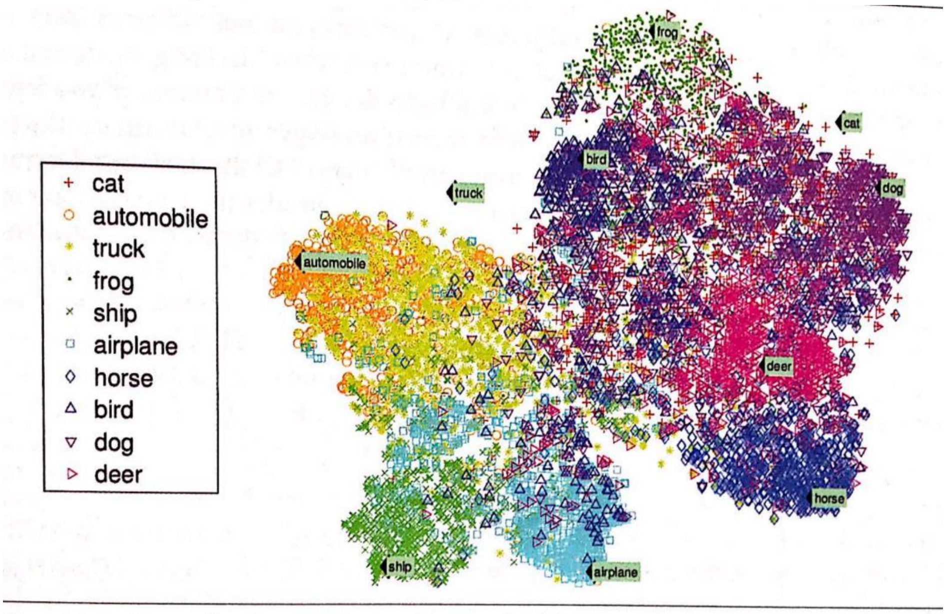


Figure 1-9. Example of a t-SNE visualization highlighting semantic clusters[3]

Regression problems:

- Forecasting your company's revenue next year, based on many performance metrics

Using Linear Regression, Polynomial Regression, regression SVM, regression **Random Forest**, NNs

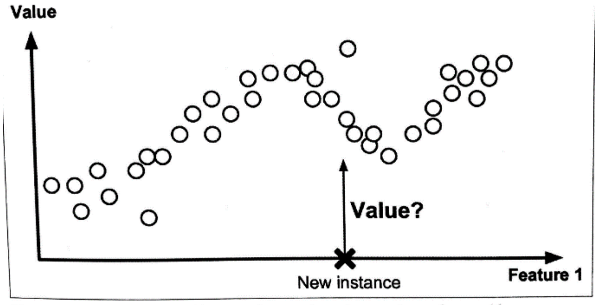


Figure 1-6. A regression problem: predict a value, given an input feature (there are usually multiple input features, and sometimes multiple output values)

Anomaly detection problems:

- Detecting credit card fraud

Using Linear Regression, Polynomial Regression, regression SVM, regression **Random Forest**, NNs

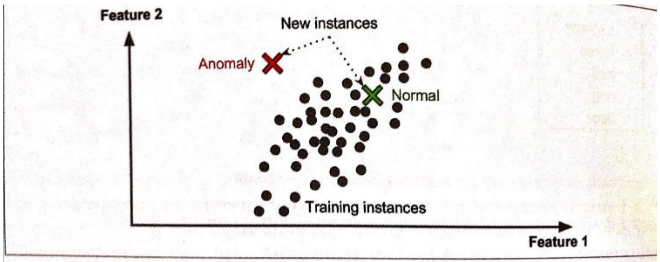


Figure 1-10. Anomaly detection

Customer segmentation problems:

- Segmenting clients based on their purchases so that you can design a different marketing strategy for each segment
- Recommending systems

Using Linear Regression, Polynomial Regression, regression SVM, regression **Random Forest**, NNs

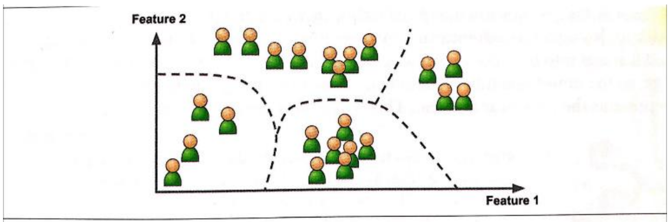


Figure 1-8. Clustering

Intelligent bots:

- Building an intelligent bot for a game

Using Reinforcement Learning

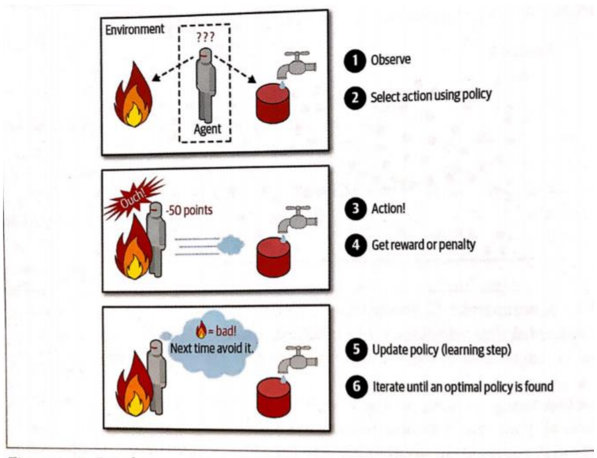


Figure 1-12. Reinforcement Learning

Modelli di Machine Learning

Decision Trees

In [87]:
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

from graphviz import Source
from sklearn.tree import export_graphviz

import os

import numpy as np

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "decision_trees"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)
os.environ["PATH"] += os.pathsep + 'D:/Program Files (x86)/Graphviz2.38/bin/'

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

In [2]:
```
iris = load_iris()
X = iris.data[:,2:] #petal width and length
y = iris.target
```

In [3]:
```
iris
```

Out[3]:
```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
```

In [4]:
```
X
```

Out[4]:
```
array([[1.4, 0.2],
       [1.4, 0.2],
       [1.3, 0.2],
       [1.5, 0.2],
       [1.4, 0.2],
       [1.7, 0.4],
       [1.4, 0.3],
       [1.5, 0.2],
       [1.4, 0.2],
       [1.5, 0.1],
       [1.5, 0.2],
       [1.6, 0.2],
       [1.4, 0.1],
       [1.1, 0.1],
       [1.2, 0.2],
       [1.5, 0.4],
       [1.3, 0.4],
       [1.4, 0.3],
       [1.7, 0.3],
```

In [5]:
```
#0 --> Iris setosa
#1 --> Iris versicolor
#2 --> Iris virginica
y
```

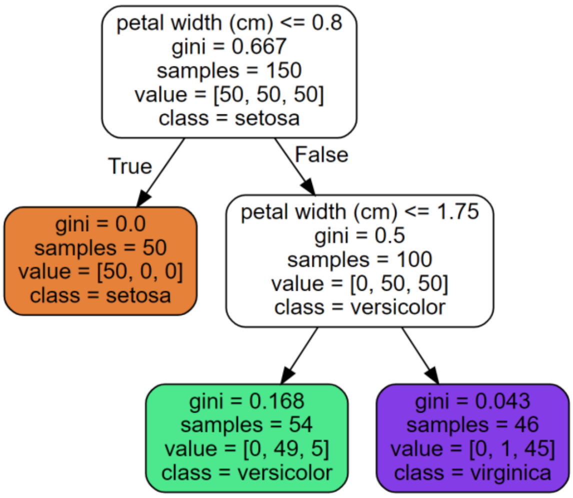
Out[5]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [6]:
```
tree_clf = DecisionTreeClassifier(max_depth=2)
tree_clf.fit(X,y)
```

Out[6]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=2, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

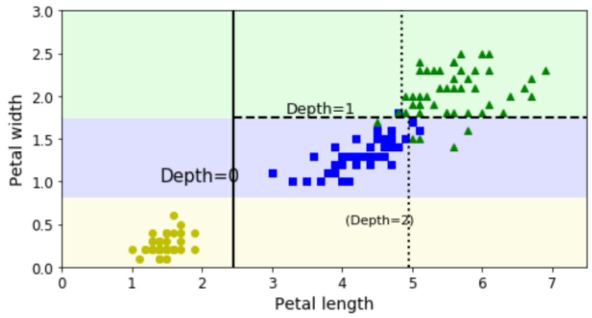
In [7]:
```
export_graphviz(
        tree_clf,
        out_file=os.path.join(IMAGES_PATH, "iris_tree.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "iris_tree.dot"))
```

Out[7]:



In [8]:
```
from matplotlib.colors import ListedColormap

def plot_decision_boundary(clf, X, y, axes=[0, 7.5, 0, 3], iris=True, legend=False, plot_training=True):
    x1s = np.linspace(axes[0], axes[1], 100)
    x2s = np.linspace(axes[2], axes[3], 100)
    x1, x2 = np.meshgrid(x1s, x2s)
    X_new = np.c_[x1.ravel(), x2.ravel()]
    y_pred = clf.predict(X_new).reshape(x1.shape)
    custom_cmap = ListedColormap(['#fafab0','#9898ff','#a0faa0'])
    plt.contourf(x1, x2, y_pred, alpha=0.3, cmap=custom_cmap)
    if not iris:
        custom_cmap2 = ListedColormap(['#7d7d58','#4c4c7f','#507d50'])
        plt.contour(x1, x2, y_pred, cmap=custom_cmap2, alpha=0.8)
    if plot_training:
        plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo", label="Iris setosa")
        plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs", label="Iris versicolor")
        plt.plot(X[:, 0][y==2], X[:, 1][y==2], "g^", label="Iris virginica")
        plt.axis(axes)
    if iris:
        plt.xlabel("Petal length", fontsize=14)
        plt.ylabel("Petal width", fontsize=14)
    else:
        plt.xlabel(r"$x_1$", fontsize=18)
        plt.ylabel(r"$x_2$", fontsize=18, rotation=0)
    if legend:
        plt.legend(loc="lower right", fontsize=14)

plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf, X, y)
plt.plot([2.45, 2.45], [0, 3], "k-", linewidth=2)
plt.plot([2.45, 7.5], [1.75, 1.75], "k--", linewidth=2)
plt.plot([4.95, 4.95], [0, 1.75], "k:", linewidth=2)
plt.plot([4.85, 4.85], [1.75, 3], "k:", linewidth=2)
plt.text(1.40, 1.0, "Depth=0", fontsize=15)
plt.text(3.2, 1.80, "Depth=1", fontsize=13)
plt.text(4.05, 0.5, "(Depth=2)", fontsize=11)
```

Out[8]: Text(4.05, 0.5, '(Depth=2)')



In [9]:
```
tree_clf.predict_proba([[5, 1.5]])
```
Out[9]: array([[0.   , 0.90740741, 0.09259259]])

In [10]:
```
tree_clf.predict([[5, 1.5]])
```
Out[10]: array([1])

Neural Networks

In [11]:
```
import tensorflow as tf
from tensorflow import keras
```

In [12]:
```
fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

In [13]:
```
X_valid, X_train = X_train_full[:5000] / 255., X_train_full[5000:] / 255.
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
X_test = X_test / 255.
```

In [14]:
```
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
               "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]
```

In [15]:
```
#immagini del dataset
n_rows = 4
n_cols = 10
plt.figure(figsize=(n_cols * 1.2, n_rows * 1.2))
for row in range(n_rows):
    for col in range(n_cols):
        index = n_cols * row + col
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(X_train[index], cmap="binary", interpolation="nearest")
        plt.axis('off')
        plt.title(class_names[y_train[index]], fontsize=12)
plt.subplots_adjust(wspace=0.2, hspace=0.5)
save_fig('fashion_mnist_plot', tight_layout=False)
plt.show()
```

Saving figure fashion_mnist_plot

In [16]:
```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

In [17]:
```
keras.backend.clear_session()
np.random.seed(42)
tf.random.set_seed(42)
```

In [18]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten (Flatten)            (None, 784)               0         
_________________________________________________________________
dense (Dense)                (None, 300)               235500    
_________________________________________________________________
dense_1 (Dense)              (None, 100)               30100     
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010      
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
```

In [20]:
```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
```

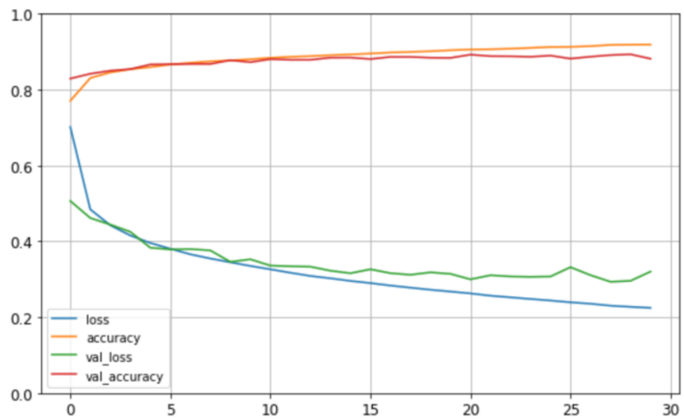
In [21]:
```
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid))
```

```
Train on 55000 samples, validate on 5000 samples
Epoch 1/30
55000/55000 [==============================] - 5s 89us/sample - loss: 0.7015 - accuracy: 0.7706 - val_loss: 0.5065 - val_acc
uracy: 0.8292
Epoch 2/30
55000/55000 [==============================] - 4s 81us/sample - loss: 0.4844 - accuracy: 0.8308 - val_loss: 0.4616 - val_acc
uracy: 0.8420
Epoch 3/30
55000/55000 [==============================] - 5s 91us/sample - loss: 0.4425 - accuracy: 0.8455 - val_loss: 0.4442 - val_acc
uracy: 0.8500
Epoch 4/30
55000/55000 [==============================] - 4s 79us/sample - loss: 0.4155 - accuracy: 0.8539 - val_loss: 0.4252 - val_acc
uracy: 0.8542
Epoch 5/30
55000/55000 [==============================] - 4s 79us/sample - loss: 0.3957 - accuracy: 0.8595 - val_loss: 0.3832 - val_acc
uracy: 0.8664
Epoch 6/30
55000/55000 [==============================] - 4s 80us/sample - loss: 0.3805 - accuracy: 0.8663 - val_loss: 0.3788 - val_acc
uracy: 0.8672
Epoch 7/30
55000/55000 [==============================] - 4s 78us/sample - loss: 0.3659 - accuracy: 0.8708 - val_loss: 0.3797 - val_acc
uracy: 0.8678
Epoch 8/30
55000/55000 [==============================] - 4s 79us/sample - loss: 0.3548 - accuracy: 0.8747 - val_loss: 0.3761 - val_acc
uracy: 0.8678
Epoch 9/30
55000/55000 [==============================] - 5s 84us/sample - loss: 0.3449 - accuracy: 0.8771 - val_loss: 0.3460 - val_acc
uracy: 0.8778
Epoch 10/30
55000/55000 [==============================] - 4s 79us/sample - loss: 0.3352 - accuracy: 0.8803 - val_loss: 0.3527 - val_acc
uracy: 0.8724
Epoch 11/30
55000/55000 [==============================] - 4s 80us/sample - loss: 0.3263 - accuracy: 0.8839 - val_loss: 0.3361 - val_acc
uracy: 0.8806
Epoch 12/30
55000/55000 [==============================] - 5s 95us/sample - loss: 0.3173 - accuracy: 0.8867 - val_loss: 0.3344 - val_acc
uracy: 0.8788
Epoch 13/30
55000/55000 [==============================] - 5s 92us/sample - loss: 0.3088 - accuracy: 0.8885 - val_loss: 0.3332 - val_acc
uracy: 0.8788
Epoch 14/30
55000/55000 [==============================] - 5s 85us/sample - loss: 0.3026 - accuracy: 0.8909 - val_loss: 0.3227 - val_acc
uracy: 0.8844
Epoch 15/30
55000/55000 [==============================] - 4s 80us/sample - loss: 0.2957 - accuracy: 0.8928 - val_loss: 0.3160 - val_acc
uracy: 0.8846
Epoch 16/30
55000/55000 [==============================] - 5s 95us/sample - loss: 0.2900 - accuracy: 0.8953 - val_loss: 0.3264 - val_acc
uracy: 0.8808
Epoch 17/30
55000/55000 [==============================] - 4s 78us/sample - loss: 0.2836 - accuracy: 0.8980 - val_loss: 0.3161 - val_acc
uracy: 0.8864
```

```
Epoch 18/30
55000/55000 [==============================] - 4s 81us/sample - loss: 0.2779 - accuracy: 0.8996 - val_loss: 0.3119 - val_acc
uracy: 0.8864
Epoch 19/30
55000/55000 [==============================] - 5s 88us/sample - loss: 0.2724 - accuracy: 0.9014 - val_loss: 0.3183 - val_acc
uracy: 0.8842
Epoch 20/30
55000/55000 [==============================] - 5s 85us/sample - loss: 0.2676 - accuracy: 0.9041 - val_loss: 0.3143 - val_acc
uracy: 0.8838
Epoch 21/30
55000/55000 [==============================] - 5s 91us/sample - loss: 0.2627 - accuracy: 0.9059 - val_loss: 0.2999 - val_acc
uracy: 0.8924
Epoch 22/30
55000/55000 [==============================] - 5s 90us/sample - loss: 0.2566 - accuracy: 0.9067 - val_loss: 0.3107 - val_acc
uracy: 0.8884
Epoch 23/30
55000/55000 [==============================] - 4s 78us/sample - loss: 0.2524 - accuracy: 0.9082 - val_loss: 0.3075 - val_acc
uracy: 0.8880- loss: 0.2
Epoch 24/30
55000/55000 [==============================] - 4s 78us/sample - loss: 0.2481 - accuracy: 0.9103 - val_loss: 0.3065 - val_acc
uracy: 0.8868
Epoch 25/30
55000/55000 [==============================] - 4s 73us/sample - loss: 0.2439 - accuracy: 0.9126 - val_loss: 0.3074 - val_acc
uracy: 0.8898
Epoch 26/30
55000/55000 [==============================] - 4s 72us/sample - loss: 0.2393 - accuracy: 0.9129 - val_loss: 0.3319 - val_acc
uracy: 0.8818
Epoch 27/30
55000/55000 [==============================] - 4s 68us/sample - loss: 0.2355 - accuracy: 0.9149 - val_loss: 0.3108 - val_acc
uracy: 0.8870
Epoch 28/30
55000/55000 [==============================] - 4s 72us/sample - loss: 0.2303 - accuracy: 0.9181 - val_loss: 0.2934 - val_acc
uracy: 0.8912
Epoch 29/30
55000/55000 [==============================] - 4s 75us/sample - loss: 0.2273 - accuracy: 0.9188 - val_loss: 0.2959 - val_acc
uracy: 0.8932
Epoch 30/30
55000/55000 [==============================] - 4s 71us/sample - loss: 0.2245 - accuracy: 0.9192 - val_loss: 0.3204 - val_acc
uracy: 0.8816
```

In [22]:
```python
import pandas as pd

pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
save_fig("keras_learning_curves_plot")
plt.show()
```

```
Saving figure keras_learning_curves_plot
```



In [89]:
```
X_new = X_test[:3]
#print(X_new)
y_proba = model.predict(X_new)
y_proba.round(2)
y_pred = model.predict_classes(X_new)
print(y_pred)
print(np.array(class_names)[y_pred])
```

```
[9 2 1]
['Ankle boot' 'Pullover' 'Trouser']
```

In [24]:
```
y_new = y_test[:3]
y_new
```

Out[24]:
```
array([9, 2, 1], dtype=uint8)
```

In [25]:
```
plt.figure(figsize=(7.2, 2.4))
for index, image in enumerate(X_new):
    plt.subplot(1, 3, index + 1)
    plt.imshow(image, cmap="binary", interpolation="nearest")
    plt.axis('off')
    plt.title(class_names[y_test[index]], fontsize=12)
plt.subplots_adjust(wspace=0.2, hspace=0.5)
save_fig('fashion_mnist_images_plot', tight_layout=False)
plt.show()
```

```
Saving figure fashion_mnist_images_plot
```

Text analysis

In [73]:
```
import pandas as pd

from collections import Counter

from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.base import TransformerMixin
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import f1_score,precision_score,recall_score,multilabel_confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.utils import class_weight
from sklearn.ensemble import RandomForestClassifier

import lime
from lime import lime_text
from lime import lime_text
from lime.lime_text import LimeTextExplainer

import tensorflow as tf
from tensorflow import keras
import tensorflow_datasets as tfds

import spacy
from spacy.lang.en.stop_words import STOP_WORDS
import string
import en_core_web_md
nlp_md = en_core_web_md.load()

import numpy as np

import xlsxwriter

import joblib

import pickle

import json
```

```
In [70]: #carico i dati
df_coded = pd.read_csv('C:/Users/lizzy/Desktop/Universita/tesi/esercizio_multilabel/data/imdb_dataframe_coded.csv')
y_completed = np.load('C:/Users/lizzy/Desktop/Universita/tesi/esercizio_multilabel/data/y_completed.npy')
labels = np.load('C:/Users/lizzy/Desktop/Universita/tesi/esercizio_multilabel/data/labels.npy')
```

```
In [74]: with open("C:/Users/lizzy/Desktop/Universita/tesi/esercizio_multilabel/imdb-spoiler-dataset/IMDB_movie_details.json") as f:
    json_file = f.read()

json_strings = [string+'}' for string in json_file.split('}\n')]

json_objs = [json.loads(string) for string in json_strings if string!='}']

df = pd.DataFrame(json_objs)

df_reduced = df[['plot_summary', 'genre']].copy()
```

```
In [75]: #codifico i generi
multilabel_binarizer = MultiLabelBinarizer()
y=multilabel_binarizer.fit_transform(df_reduced['genre'])
```

```
In [76]: xtrain, xtest, ytrain, ytest = train_test_split(df_coded['plot_summary'], y_completed, test_size=0.2, random_state=10)
```

```
In [77]: #carico il model
one_vs_rest_random_forest = joblib.load('C:/Users/lizzy/Desktop/Universita/tesi/esercizio_multilabel/models/random_forest_model.
sav')
```

```
In [78]: #tree
tfvectorizer = TfidfVectorizer(max_df=0.8, max_features=10000)
tfvectorizer.fit(xtrain,ytrain)
pipe_tree = make_pipeline(tfvectorizer, one_vs_rest_random_forest)
```

```
In [79]: explainer = LimeTextExplainer(class_names = labels)
```

```
In [80]: idx_t = 3
```

```
In [81]: exp = explainer.explain_instance(xtrain[idx_t], pipe_tree.predict_proba, num_features=12, top_labels=3)
```

```
In [82]: y_pred_prob = pipe_tree.predict_proba(xtrain)
```

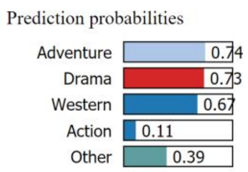
```
In [83]: t = 0.5 # threshold value
y_pred_new = (y_pred_prob >= t).astype(int)
len(ytrain[0])

Out[83]: 21
```

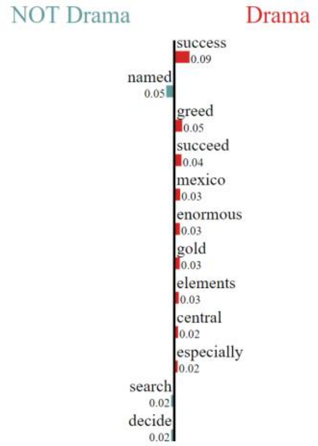
```
In [84]: print("Testo")
print([item for item in xtrain][idx_t])
print("True label")
print(multilabel_binarizer.inverse_transform(ytrain)[idx_t])
print("Predicted label")
print(multilabel_binarizer.inverse_transform(y_pred_new)[idx_t])

Testo
black woman chosen stage reno lounge singer lead girl trio chooses arranges music choreographs shows wisecracking showy woman l
oved music current job hired married lover sing casino lounge learns vince true business gangster walks killing employees wrong
ed witness murder deloris goes run police lt long running operation evidence vince bars murder proverbial nail vince coffin vin
ce contract deloris life prevent testifying eddie hide trial eddie chooses poor catholic parish
True label
('Comedy', 'Crime', 'Family')
Predicted label
('Comedy', 'Crime', 'Family')
```

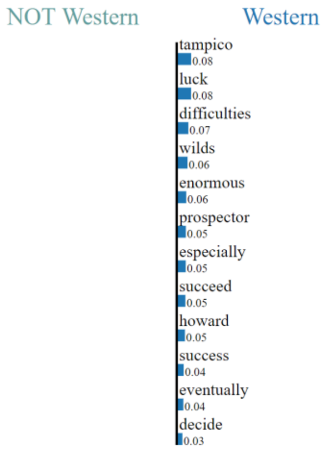
```
In [85]: for idx in exp.available_labels():
    print ('Explanation for class %s' % list(labels)[idx])
    print ('\n'.join(map(str, exp.as_list(label=idx))))
    print ()

Explanation for class Adventure
('search', 0.10450936506466124)
('elements', 0.08725116194524095)
('finding', 0.06575113739139284)
('luck', 0.05643634851798891)
('meet', 0.040963975826599705)
('enormous', 0.040276883307449274)
('disaster', 0.03829687272331598)
('join', 0.03803927039757325)
('mexico', 0.03552572475523216)
('named', 0.033029106917141386)
('succeed', 0.029530982412451435)
('1925', 0.027101412491666352)

Explanation for class Drama
('success', 0.08882851015722856)
('named', -0.04643117323474262)
('greed', 0.04623257533332577)
('succeed', 0.042058152324959464)
('mexico', 0.03456060607126048)
('enormous', 0.032636704909045675)
('gold', 0.03189787554114824)
('elements', 0.026286572209421006)
('central', 0.022086895632081414)
('especially', 0.019923648481215625)
('search', -0.018235854164668688)
('decide', -0.01823453736856582)
```

```
Explanation for class Western
('tampico', 0.0838597687194699)
('luck', 0.08351129358169741)
('difficulties', 0.06773556984520424)
('wilds', 0.0629276474465334)
('enormous', 0.05527658853516896)
('prospector', 0.053247101699147426)
('especially', 0.052476641074379216)
('succeed', 0.05037504444392739)
('howard', 0.047761742112329045)
('success', 0.04044231544301416)
('eventually', 0.03855527779649293)
('decide', 0.03294314387395319)
```

In [86]:
```
exp.show_in_notebook(text=xtrain[idx_t], labels=exp.available_labels())
```







Text with highlighted words

luck tampico mexico 1925 meet grizzled prospector named howard decide join search gold wilds central mexico enormous difficulties eventually succeed finding gold bandits elements especially greed threaten turn success disaster